Académie de Montpellier
Université Montpellier II
— Sciences et Techniques du Languedoc —

Thèse

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier

Spécialité	:	**Informatique**
Formation Doctorale	:	**Informatique**
École Doctorale	:	**Information, Structures, Systèmes**

Impact de la contrainte d'incompatibilité sur la complexité et l'approximation des problèmes d'ordonnancement en présence de tâches-couplées

par

Gilles Simonin

Soutenue le (01/12/2009), devant le jury composé de :

Evripidis Bampis, Rapporteur
Nadia Brauner Vettier, Rapporteur
Claire Hanen, Examinateur
Christophe Paul, examinateur
Rodolphe Giroudeau, Directeur de thèse
Jean-Claude König, Directeur de thèse

À mes parents,
et à Céline.

Remerciements :

La plus grande difficulté avec les remerciements, c'est d'essayer de penser à tout le monde. Mais durant ces trois années, il y a tellement de personnes qui m'ont aidé à traverser cette épreuve, que je ne peux pas tous les citer. Et je m'excuse donc par avance pour les noms qui ne sont pas cités.

Pour commencer, je tiens à remercier mes directeurs de thèses Jean-Claude König et Rodolphe Giroudeau, sans qui cette thèse n'aurait pas pu exister, progresser et aboutir. Je les remercie de m'avoir toujours laissé cette part d'autonomie sur les problèmes que je voulais traiter et pour m'avoir fouetté afin que je travaille plus vite et plus efficacement ! Cette dernière n'aurait pas vu le jour sans la confiance, la patience, et la générosité qu'ils ont su m'accorder. Ce fut un réel plaisir de partager toutes ces heures à vos côtés sur des problèmes passionnants. J'espère que nous continuerons à travailler ensemble.

Je remercie Nadia Brauner Vettier et Evripidis Bampis qui m'ont fait l'honneur d'être rapporteurs de cette thèse. J'ai grandement apprécié le regard critique qu'ils ont porté à mon travail, et toutes les remarques pertinentes que Nadia a pu me faire au fur et à mesure que je finissais le dernier chapitre. Merci également à Claire Hanen et Christophe Paul d'avoir accepté de faire parti de ce jury en tant qu'examinateur.

Je désire remercier les membres de l'équipe Algorithmique et Performances des Réseaux du LIRMM à Montpellier, au sein de laquelle j'ai effectué cette thèse. Je ne pouvais y espérer un meilleur accueil que celui qui m'a été octroyé. Je tiens à remercier tout particulièrement Anne-Elisabeth et Alain pour leur collaboration sur la partie Stochastique de ma thèse et leur aide en général durant ces trois années. Merci à Vincent pour ces nombreuses heures à discuter de tout et de rien et notamment de "SG1", ca va me manquer... Tak'matak.

Il y a trop de gens à remercier au LIRMM pour tous les citer, mais certain ont plus compté que d'autre. Merci donc à Stéphan Thomasse et Stéphane Bessy pour avoir bien voulu répondre à toutes mes questions de graphe durant ces trois ans, et il y en a eu ! Merci à Marianne Huchard pour avoir accepté d'être ma tutrice de monitorat, de faire partie de mon comité de thèse, et de sa gentillesse au quotidien.

Une thèse a beau être une aventure en solitaire, il existe tout un troupeau de doctorants qui sont dans la même galère, et qui font qu'on se sent moins seul... Travailler à leurs cotés fut une expérience enrichissante et les nombreux échanges scientifiques que nous avons eu m'ont permis d'approfondir mes connaissances dans de nombreux domaines. Je remercie mes co-bureaux de m'avoir souvent donné la possibilité de travailler seul et au calme. Un merci particulier à ceux qui ont fait des pauses avec moi durant ces trois années, moments de détente sans lesquels les journées auraient été vraiment trop longues ! Merci donc à : Julien, Xavier, Ben, Flop, Paola, Jre, Raluca, Lisa, Thomas, Binh-Minh, Christophe,

Marwane, etc... Merci également aux thésards du département mathématique avec qui j'ai passé de bons moments, en particulier : Thomas, Soffana, Chloé, Benoit, Nadia, Guillaume, etc...

Dans les moments détentes, je remercie également tous les membres du séminaire du vendredi soir, merci à Rémi et Guillaume d'avoir souvent prêté leur bureau pour nos sessions, et pour avoir stocké les actes durant de long mois ! Puis un grand merci à tous ceux d'IRC pour avoir été là quasiment tout le temps, merci donc à : Lartick, Valek, Shura, Fannoche, Jokerman, Bastien, Leander, Noemie, Psykoo, Ercete et enfin à notre bot préféré.

À titre plus personnel je remercie toute ma famille à commencer par mes parents, pour avoir toujours été là, pour m'avoir donné la chance de faire de longue étude, pour leur soutien sans limite durant toute ma scolarité et plus particulièrement ces trois dernières années. Merci à mes frères et soeur pour leurs encouragements et pour les bons moments passés ensemble quand on peut se retrouver, merci à Olivier en particulier pour tous ses conseils en rapport à la thèse et la recherche en général.

Si j'ai pu faire cette thèse sans devenir fou, c'est avant tout grâce aux activités extérieures et aux amis qui sont toujours là pour me changer les idées. Un grand merci à ceux avec qui j'ai fait du théâtre et qui m'ont accompagné durant toutes ces années à Montpellier, ils sont vraiment trop nombreux pour tous les citer alors je remercierai les différentes compagnies qui m'ont accueilli et formé : Le TAUST qui m'a permis de connaître le théâtre et apprendre à jouer sur scène, un merci particulier à Laurence Vigné qui a changé ma vision du théâtre et qui m'a permis de trouver mon style. Merci à Jicé, Céline et Karl pour leur beau cadeau de fin de thèse, "it was legendary !"

Puis tous ceux de la compagnie Auguste singe avec qui j'ai pu expérimenter tout ce qui me plaisait au théâtre, en particulier merci à Matthieu qui est la personne avec qui j'ai pris le plus de plaisir à jouer sur scène, il m'a permis d'écrire et jouer dans un vrai théâtre ma première pièce, de me lancer dans l'écriture de sketchs, de numéro de clown, et pour avoir été là quand ça n'allait pas... thanks dude, nos routes se séparent mais j'espère pouvoir rejouer avec toi dans l'avenir. Et également un grand grand merci à ma présidente chérie préférée Caroline, qui m'a fait les bords de mon pantalon la veille de ma soutenance, et grâce auquel j'ai pu présenter ma thèse avec classe et sérénité ! T'es la meilleure Caro !

Et enfin tous ceux de la compagnie du capitaine, merci à Julien pour m'avoir donné la chance de connaître l'expérience d'une compagnie professionnelle, et pour avoir supporté ma folie durant ces trois ans. Il y a bien d'autre gens que j'ai rencontré et avec qui j'ai joué et qui ne font pas parti de ces troupes ou associations, et je les remercie également.

Je conclurai en remerciant l'ensemble de mes amis non théâtreux pour leur soutien, et les moments de détente que j'ai pus partager avec eux entre deux séances de travail, et particulièrement : Guillaume, Émilie, Kurgan, Éric, Yannick, Cécile, Marie, Marie-céline, Thomas, Gaëlle, Axelle, Vincent, Éric, Maxime, Fred, Julie, Julien, Nicolas, Petrutza, etc...

Et puis deux personnes que je mettrai à part, qui m'ont permis d'être là où je suis aujourd'hui. Dans la vie il y a des gens qui nous touchent plus que d'autre et qui seront toujours là pour nous. En premier lieu merci à Benoit d'avoir été le meilleur co-bureau que je pouvais rêver et un ami hors pair sur qui on peut compter dans toutes les épreuves de la vie. Mes deux premières années de thèse n'auraient pas été si drôle, et l'envie d'aller au labo pas aussi grande s'il n'avait pas été là. Merci à lui pour toutes nos discussions scientifiques et surtout celles non scientifiques, pour tous ce qu'on a partagé ensemble et bien plus encore. Il a toujours été là quand il fallait, malgré la distance.

Et enfin merci à Céline, je ne saurai jamais assez la remercier de ce qu'elle a fait pour moi. Cette thèse n'aurait pas pu se finir dans les temps si elle n'avait pas été là. C'est amusant que ce soit la personne la plus loin géographiquement de moi qui est été la plus proche durant ces six derniers mois, merci à elle pour m'avoir encouragé, changé les idées, forcé à me bouger pour ne penser qu'à ma thèse. Comme Benoit elle a toujours été là pour moi malgré la distance, et malgré les horaires de décalage ! C'est quand on est au plus mal qu'on fait toujours les plus belles rencontres. Merci pour tout.

Voilà, c'était long mais il y a eu tellement de personne qui ont compté et qui m'ont aidé que je leur devais bien ça. Et puis de cette manière vous vous êtes échauffé pour lire ma thèse. Elle n'est pas si longue que ça mais il faut du courage, alors il me reste plus qu' à vous souhaiter une bonne lecture et "Que la force soit avec vous" !

Table des matières

Première partie

Présentation

Chapitre 1

Introduction

La phrase de Milan Kundera «La vitesse est la forme d'extase dont la révolution technique a fait cadeau à l'homme» est à l'image du $XX^{ième}$ siècle, depuis le Taylorisme en passant par le Fordisme, puis le Toyotisme, l'homme n'a de cesse cherché qu'efficacité et rapidité. L'informatique et la robotique ont joué un rôle important à cet effet, une machine est plus rapide et précise qu'un être humain, et elle n'a pas besoin de faire de pause. La robotique s'est également imposée pour réaliser des tâches dangereuses et périlleuses à la place de l'homme, comme par exemple la peinture des carrosseries automobiles en atmosphère de vapeurs toxiques. Les premiers robots n'avaient aucun pouvoir de décision, et devaient répéter inlassablement les tâches qui leur incombaient. Mais la miniaturisation des composants et la réduction des coûts de production ont permis de diversifier l'utilisation de la robotique, équipant les robots de capteurs pour sonder leur environnement et d'unité de calcul pour analyser les situations et prendre des décisions.

Dans la volonté grandissante de remplacer l'homme par des robots pour des tâches dangereuses, la robotique a été amenée à évoluer vers des modèles plus autonomes. Le robot n'a plus à effectuer une unique tâche primaire de manière répétitive mais acquiert la possibilité de pouvoir décider et interagir avec un environnement souvent inconnu et en constante évolution. L'exploration sous-marine est un parfait exemple d'application nécessitant la conception de véhicules robotisés. Le besoin d'opérer dans les eaux de plus en plus profondes et de réduire les coûts, amène les recherches à se concentrer sur l'élaboration de véhicules autonomes (par exemple les AUV - Autonomous Underwater Vehicles) capables de se déplacer seuls et de mener à bien des tâches qui ne peuvent être assurées par un opérateur humain. Ce besoin d'autonomie dans un milieu en constante évolution requiert de la part du véhicule une certaine capacité à pouvoir, à chaque instant, évaluer son propre état et celui de son environnement, les combiner avec la mission qui lui a été confiée et prendre une décision cohérente.

Des roboticiens du LIRMM travaillent sur des véhicules sous-marins autonomes, et plus explicitement sur une torpille nommée TAIPAN.

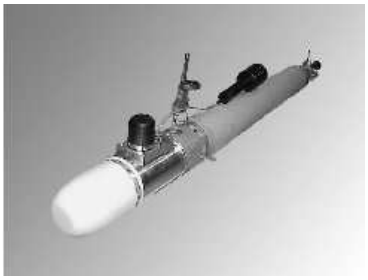


Fig. 1.1 – La torpille TAIPAN

Ce véhicule sous-marin présente plusieurs niveaux d'autonomie. Autonomie de déplacements grâce à des batteries qui lui permettent de réaliser des parcours de l'ordre de 17 kilomètres. Autonomie pour la navigation grâce à son pilote automatique, TAIPAN peut se mettre en plongée avec la possibilité de conserver son assiette constante. TAIPAN calcule en permanence ses déplacements à l'aide des informations délivrées par des capteurs d'accélération, de vitesse et d'inclinaison, et enfin un capteur de pression lui permettant de connaître sa profondeur d'immersion.

Fig. 1.2 – La torpille TAIPAN en plongée

L'utilité de TAIPAN est d'être capable d'embarquer des capteurs de mesures physico-chimiques comme la température, la salinité, la conductivité par exemple, ou d'embarquer des capteurs acoustiques pour cartographier les fonds sous marins. Les données collectées sont mémorisées dans TAIPAN.

Un exemple concret d'utilisation de TAIPAN est l'inspection d'un pipeline, ce travail consiste à déterminer deux types d'accidents : le recouvrement du pipeline par des sédiments ou du sable, ou au contraire de déterminer les zones où le pipeline ne repose plus sur le fond. Il est alors nécessaire de photographier les points d'ancrage du pipeline. Tout cela, TAIPAN doit pouvoir le réaliser de façon automatique et doit avoir une stratégie de collecte afin de minimiser la quantité de données à mémoriser.

Fig. 1.3 – Inspection d'un pipeline

Pour des raisons d'économie d'énergie et d'argent, de simplicité architecturale et de choix personnel, les roboticiens du LIRMM ont doté la torpille d'un mono processeur. Ce processeur servira à gérer les différents calculs pour activer les capteurs et réaliser des traitements. La construction d'une torpille et sa mise en œuvre coûtant énormément d'argent et de temps, leur volonté est d'exploiter ce processeur au maximum afin de rentabiliser l'utilisation de la torpille en fond sous-marin, ce qui se traduit par la volonté d'exécuter une multitude de tâches en un minimum de temps, selon l'hétérogénéité au niveau des tâches et des différentes contraintes.

TAIPAN doit exécuter les différentes tâches à intervalle de temps régulier. Nous pouvons classer en deux catégories les tâches à exécuter : les tâches d'acquisition, qui servent à récupérer des informations, comme par exemple le calcul de la salinité de l'eau, puis les tâches de traitement qui vont réaliser différents calculs à partir des tâches d'acquisition exécutées. Elles peuvent être interrompues à n'importe quel moment et être reprises plus tard, c'est ce que nous appelons des tâches préemptives.

L'ensemble des tâches d'acquisition et de traitement forme plusieurs groupes de tâches reliées entre elles, ces groupes de tâches devront s'exécuter avant leur date limite d'exécution, et de façon périodique.

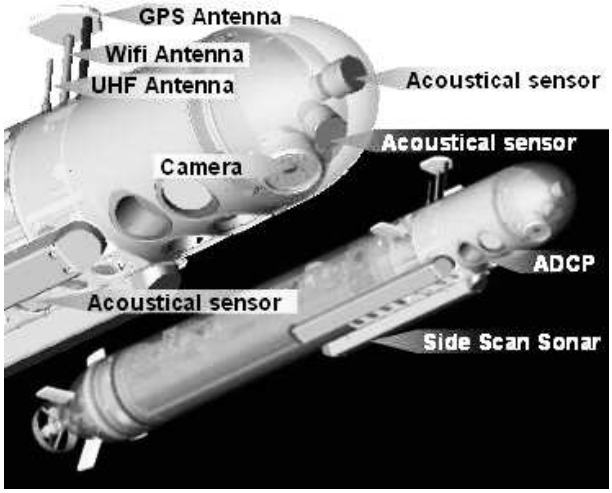


Fig. 1.4 – Position des différents capteurs

Les tâches n'ont pas toutes le même temps d'exécution, ceci est dû à leurs différentes spécificités, par exemple le calcul de la meilleure direction à prendre pour la torpille nécessitera plus de temps de calcul que pour la mesure de la température.

Les tâches d'acquisition sont composées de deux sous-tâches différentes, la première va consister à préparer le capteur pour l'envoi d'un écho dans un but bien précis, et la deuxième va réceptionner les informations collectées par l'écho et les stocker dans un tampon mémoire après qu'il soit revenu. Entre ces deux sous-tâches, il y a un temps d'attente incompressible et indilatable durant lequel l'écho part, se déplace, puis revient au capteur.

Comme nous l'avons vu précédemment, le désir des roboticiens est d'économiser les temps d'inactivité du processeur pour optimiser l'utilisation de la ressource et avoir un meilleur rendement du processeur embarqué. Il va donc être naturel d'utiliser les temps d'attente incompressibles et indilatables des tâches d'acquisition pour exécuter d'autres tâches. Ainsi la torpille pourra réaliser un maximum de tâches en un minimum de temps.

Cependant certains capteurs (la plupart du temps des capteurs acoustiques adjacents), ne peuvent effectuer leur mesure en même temps au risque de produire des interférences qui se répercuteront en erreurs de mesures sur les données acquises. Ainsi certaines de ces tâches ne doivent pas être exécutées en même temps. Pour gérer cette contrainte, les roboticiens utilisent un graphe dit de compatibilité pour représenter par des arêtes les tâches qui pourront être exécutées en même temps.

Notre collaboration avec les roboticiens est née de leur besoin de résoudre de manière algorithmique leur problème avant de procéder à des tests réels, et de proposer une étude théorique sur la complexité et l'approximation des différents problèmes existants. Notre travail a consisté dans un premier temps à modéliser le problème de la torpille à l'aide de la théorie de l'ordonnancement et des graphes. Cette modélisation a nécessité une recherche des différentes structures déjà existantes dans la littérature afin de représenter au mieux nos données. Naturellement la théorie de l'ordonnancement s'est imposé comme étant le meilleur support pour étudier le problème général. Notre travail a donc consisté à utiliser la théorie de l'ordonnancement dans le but d'exécuter un nombre maximum de tâches en un minimum de temps. L'hétérogénéité des paramètres à faire varier et des contraintes particulières de notre problème nous a conduit à étudier une multitude de nouveaux problèmes.

Dans un second temps, nos travaux ont porté sur une étude déterministe théorique, via l'étude de la complexité et de l'approximation, des problèmes d'ordonnancement rencontrés selon la valeur des différents paramètres. L'analyse de la complexité a consisté à classifier les différents problèmes étudiés selon leur complexité, et l'étude de l'approximation a porté sur une analyse au pire cas, cherchant la performance relative la plus petite possible.

La première partie de ce mémoire sera consacrée à la présentation de prérequis et diverses notations, puis à une modélisation du problème de la torpille vers un problème d'ordonnancement équivalent, et enfin à un état de l'art sur ce type de problème en théorie de l'ordonnancement. Les différentes contraintes propres à la torpille entraînent l'utilisation de graphes, et toute analyse passera par la théorie des graphes, nous présenterons donc les bases de la théorie des graphes utilisés dans ce mémoire et référencerons les différents problèmes connus utilisés.

Dans une deuxième partie, nous étudierons la résolution algorithmique et la complexité d'un grand nombre de problèmes théoriques classés selon la présence ou non de certaines contraintes. L'étude portera à chaque fois sur un ensemble de tâches d'acquisition en fonction de leurs paramètres, ces problèmes seront étudiés selon la présence ou non des tâches de traitement et la contrainte d'incompatibilité. Pour chaque problème, nous étudierons sa complexité et donnerons des algorithmes donnant en temps polynomial soit l'optimal soit une borne d'approximation.

Enfin la dernière partie portera sur une étude expérimentale, où nous présenterons les résultats obtenus en moyenne à partir des simulations des différents algorithmes étudiés. Les résultats théorique portant sur une étude au pire cas, nous observerons par des simulations en moyenne l'efficacité réelle de ces algorithmes. Puis nous comparerons ces données par rapport aux valeurs théoriques obtenues, nous permettant de mieux observer l'impact de la présence ou non de certaines contraintes et paramètres.

Chapitre 2

Préliminaires

2.1 Pré requis et notations

Dans cette première section, nous allons donner les notions élémentaires requises pour une bonne lecture de ce document.

2.1.1 Notations de la théorie des ensembles

Nous commençons par préciser les notations de la théorie des ensembles utilisées tout au long du manuscrit.

Le cardinal d'un ensemble A est noté $|A|$. La notation $\emptyset$ désigne l'ensemble vide.

On note $\mathbb{N}$ l'ensemble des entiers naturels, et $\mathbb{N}^*$ l'ensemble des entiers naturels strictement positifs.

La notation $\boldsymbol{[a, b]}$ désigne l'ensemble des entiers compris entre a et b.

Les éléments d'un ensemble A, de cardinal k, seront toujours représentés de la manière suivante $A = (x_1, x_2, \ldots, x_k)$ (la lettre x n'est qu'un exemple ici).

On note $\boldsymbol{\{x|p(x)\}}$ l'ensemble formé des éléments x qui vérifient la propriété $p(x)$. Lorsque les éléments sont indicés, nous préférons la notation $\boldsymbol{\{x_i\}_{i\in[1,k]}}$ pour désigner l'ensemble comprenant les éléments $x_1, x_2, \ldots, x_k$.

Nous utiliserons souvent ces trois opérations ensemblistes :

- L'**union** de deux ensembles A et B, noté $\boldsymbol{A \cup B}$, représente l'ensemble des éléments appartenant à A **ou** B.
- L'**intersection** de deux ensembles A et B, noté $\boldsymbol{A \cap B}$, représente l'ensemble des éléments appartenant à A **et** B.
- La **différence** de deux ensembles A et B, noté $\boldsymbol{A/B}$, représente l'ensemble des éléments de A qui n'appartiennent pas à B.

2.1.2 Éléments de la théorie des graphes

La théorie des graphes est apparue pour la première fois en 1735 grâce au mathématicien suisse Leonhard Euler, lorsqu'il présenta le problème des sept ponts de Königsberg à l'Académie de Saint Pétersbourg. La théorie des graphes est une branche des mathématiques qui, historiquement, s'est développée au sein de disciplines diverses telles que la chimie (modélisation de

structures), la biologie (génome), les sciences sociales (modélisation des relations) ou en vue d'applications industrielles.

Un graphe permet de représenter simplement la structure et les connexions d'un ensemble d'entités, en exprimant les relations ou dépendances entre ses éléments (réseaux ferroviaire, arbre généalogique, coloration d'une carte). À partir de cette forte base mathématique, les graphes sont devenus des structures de données puissantes pour l'informatique.

Ainsi en 1960, le français Claude Berge pose les bases de la théorie moderne des graphes. Les premières définitions formelles n'ont pas été très utilisées dû à leur lourdeur, de nombreux éléments de la théorie des graphes peuvent être définis de différentes manières. Ceci peut avoir comme conséquence l'apparition de difficultés de compréhension. Nous proposons dans cette section les principales définitions de la théorie des graphes afin de prévenir toute ambiguïté. Ces définitions prévalent dans ce manuscrit sur les notations couramment utilisées en littérature.

Graphes non orientés

Définition 2.1.1 (Un graphe, un graphe non orienté) :
Un graphe (ou graphe non orienté) $\boldsymbol{G}$ *est défini par un couple* $\boldsymbol{(V, E)}$*, où* V *est un ensemble fini non vide et* E *une relation binaire commutative sur* V*. L'ensemble* V *est appelé l'ensemble des* ***sommets*** *de* G*, et* E *l'ensemble des* ***arêtes*** *de* G*.*

La figure 2.1(a) montre un graphe non orienté de 6 *sommets.*

Nous utiliserons parfois la notation fonctionnelle $\boldsymbol{V(G)}$ pour désigner l'ensemble des sommets d'un graphe G, et $\boldsymbol{E(G)}$ l'ensemble de ses arêtes. Sauf précision nous désignerons par n et m les cardinaux respectifs de $V(G)$ et $E(G)$. Pour un graphe G donné, nous définissons le vocabulaire suivant :

- Si $\boldsymbol{\{a, b\}}$ est une arête de G, on dit que $\{a, b\}$ est **incidente** aux sommets a et b.
- Deux sommets a et b sont **adjacents** dans G si l'arête $\{a, b\}$ appartient à $E(G)$.
- Pour un sommet a, l'**ensemble des voisins** de a dans G, noté $\boldsymbol{N_G(a)}$, est l'ensemble des sommets qui lui sont adjacents.
- Le **degré** d'un sommet a de G, noté $\boldsymbol{d_G(a)}$, est le nombre de sommets qui lui sont adjacents, c'est à dire $|N_G(a)|$.
- Nous notons respectivement $\boldsymbol{\delta(G)}$ et $\boldsymbol{\Delta(G)}$ le plus petit et le plus grand degré parmi les sommets de G.

En l'absence d'ambiguïté, le G disparaît des notations : $\boldsymbol{V, E, N(a), \Delta, \delta}$.

Définition 2.1.2 (Un sous-graphe et un sous-graphe induit) :
Soit $G = (V, E)$ *un graphe, un* ***sous-graphe*** *de* G *est un graphe* $G' = (V', E')$ *tel que :*

- $V' \subseteq V$
- $E' \subseteq E \cap \{\{a, b\} | a, b \in V'\}$

La figure 2.1(c) montre un exemple de sous-graphe induit.

Si $E' = E \cap \{\{a, b\} | a, b \in V'\}$*, alors* G' *est le* ***sous-graphe*** *de* G ***induit*** *par* V'*.*

Définition 2.1.3 (Un graphe partiel) :
Soit $G = (V, E)$ *un graphe, un* ***graphe partiel*** *de* G *est un graphe* $G' = (V, E')$ *avec* $E' \subseteq E$*.*

Définition 2.1.4 (Une chaîne) :
Une ***chaîne*** C *est un graphe dans lequel :*

- $V(C) = \{x_0, x_1, \ldots, x_k\}$ *avec* $k \geq 0$*,*

– $E(C) = \{\{x_i, x_{i+1}\} |\ i \in [0, k-1]\}$.

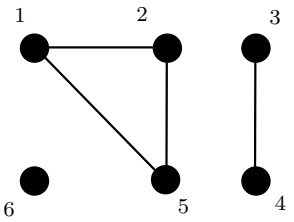
Une chaîne d'un graphe G est donc un sous-graphe de G. Pour une chaîne C donnée :

– *Les **extrémités** de C sont les sommets x_0 et x_k. Nous dirons que C **relie les sommets** x_0 et x_k.*
– *Les sommets x_1 à x_{k-1} sont les **maillons internes** de C.*
– *La **longueur** de C est égale à son nombre d'arêtes $|E(C)|$ (ici k).*

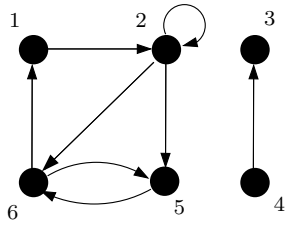
Définition 2.1.5 (Un cycle) :
*Un **cycle** P est un graphe dans lequel :*

– $V(P) = \{x_0, x_1, \ldots, x_k\}$, *avec* $k \geq 0$,
– $E(P) = \{\{x_i, x_{i+1}\} |\ i \in [0, k-1]\} \cup \{x_k, x_0\}$

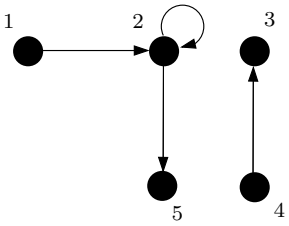
Un cycle d'un graphe G est donc une chaîne de G ayant comme extrémités le même sommet x_0. La longueur d'un cycle est égale à son nombre d'arêtes $|E(P)|$ (ici $k+1$).



(a) Un graphe non orienté



(b) Un graphe orienté



(c) Le sous-graphe du graphe (*b*) induit par les sommets $\{1, 2, 3, 4, 5\}$

Fig. 2.1 – Illustrations d'un graphe non orienté, orienté, et un sous graphe induit

Définition 2.1.6 (Un graphe connexe, une composante connexe) :
*Un graphe non vide $G = (V, E)$ est dit **connexe** si toute paire de ses sommets est reliée par une chaîne de G.*
*Une **composante connexe** de G est un sous-graphe connexe de cardinalité maximale.*

Un graphe connexe possède une seule composante connexe.

Définition 2.1.7 (Une forêt, un arbre) :
*Une **forêt** est un graphe sans cycle (acyclique).*
*Une forêt connexe est appelée un **arbre**.*

Une forêt est donc un graphe dont les composantes sont des arbres.
La figure 2.2(a) montre un arbre de 10 *sommets.*

Définition 2.1.8 (Un couplage) :
*Soit $G = (V, E)$ un graphe, un **couplage** M dans G est un ensemble d'arêtes non adjacentes. Un sommet est **couvert** par M si il est incident à une arête du couplage, sinon le sommet est dit **isolé**.*

Graphes orientés

Définition 2.1.9 (Un graphe orienté) :
*Un **graphe orienté** $G = (V, E)$ est un graphe dont les liens entre deux sommets sont orientés.*

Les éléments de R sont appelés des ***arcs****.*

La figure 2.1(b) montre un graphe orienté de 6 *sommets.*

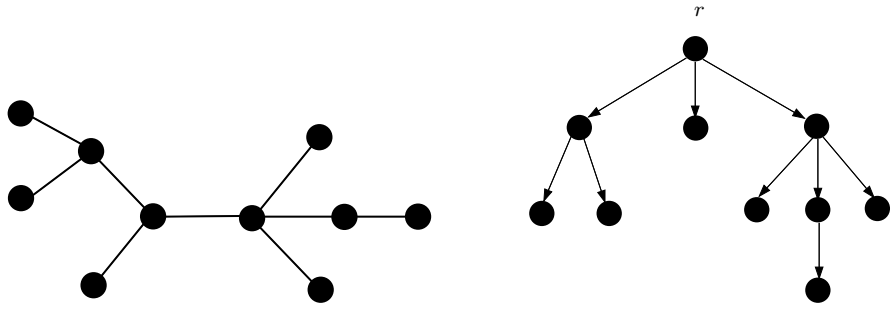
Définition 2.1.10 (Un chemin) :
Un ***chemin*** *P est un graphe orienté dans lequel :*
- $V(P) = \{x_0, x_1, \ldots, x_k\}$ *avec* $k \geq 0$,
- $E(P) = \{\{x_i, x_{i+1}\} |\ i \in [0, k-1]\}$.

Un chemin d'un graphe orienté G est donc un sous-graphe orienté de G. Pour un chemin P donné :
- L'**origine** et la **destination** de P, notées $ori(P)$ et $dest(P)$, sont respectivement les sommets x_0 et x_k.
- La **longueur** de P est égale à son nombre d'arcs $|E(P)|$ (ici k).

Définition 2.1.11 (Une arborescence) :
Une ***arborescence****, ou arbre enraciné, est un arbre avec un sommet distingué, appelé la* ***racine****. Tout sommet de degré supérieur à* 1 *est nommé* ***noeud****, les autres sont appelés* ***feuilles****. Les arcs sont alors orientés de la racine vers les feuilles.*

On représentera une arborescence avec la racine en haut du dessin et les feuilles en bas. Pour une arborescence donnée T de racine r :
- Tout noeud b se trouvant sur l'unique chemin de r à a est appelé **ancêtre** de a. a et b sont liés par une **relation de descendance**.
- Si b est un ancêtre de a, alors a est un **descendant** de b.
- Le noeud b est appelé **père** de a si b est ancêtre de a et lui est adjacent (nous noterons $pre(a) = b$). Et inversement a est appelé le **fils** de b.
- La racine r est le seul noeud de T sans père.
- La **profondeur** d'un noeud a dans une arborescence de source r est la longueur du chemin allant de r vers a, c'est à dire son nombre d'ancêtres. Nous la notons $\boldsymbol{prof_T(a)}$.
- La **hauteur** de T est la plus grande des profondeurs de ses feuilles.

La figure 2.2(b) montre une arborescence définie en enracinant l'arbre de la figure 2.2(a) sur le sommet r.



(a) Un arbre (b) Une arborescence de racine r

Fig. 2.2 – Illustrations d'un arbre et d'une arborescence

2.1.3 Éléments de la théorie de l'ordonnancement

La théorie de l'ordonnancement a commencé à être prise en compte au début du siècle dernier avec le travail d'Henry Gantt et d'autres précurseurs. Toutefois, il aura fallu du temps pour que les premières publications scientifiques apparaissent dans les année 1950 avec Smith, Johnson et Jackson [64]. Puis à partir des années 1960 la théorie de l'ordonnancement explose, une grande quantité de problèmes émerge, et durant 30 ans une quantité considérable d'études théoriques a été réalisée dans le domaine de l'ordonnancement déterministe.

La théorie de l'ordonnancement apporte précisément une réponse à la problématique de la torpille. Il s'agit d'une branche de la recherche opérationnelle qui s'intéresse au calcul de dates optimales d'exécution de tâches. L'ordonnancement consiste à allouer des ressources à des tâches tout en cherchant à optimiser un ou plusieurs objectifs. Les ressources peuvent représenter des machines dans une usine, des pistes dans un aéroport, des processeurs dans un ordinateur, etc. Les tâches peuvent être assimilées à des processus de production, des décollages et atterrissages d'avions, des programmes informatiques à exécuter, etc. Les tâches peuvent avoir différentes formes, priorités, contraintes de temps pour être exécutées. Et enfin les objectifs peuvent varier selon les problèmes modélisés et le but recherché.

Cette grande variété des problèmes d'ordonnancement a motivé l'introduction d'une notation synthétique pour classifier les schémas. Nous reprenons la notation proposée par Graham et al. [37] et par Blażewicz et al. [12].

Notation des problèmes

Pour caractériser un problème d'ordonnancement, nous avons besoin d'introduire trois paramètres $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$. Ces trois paramètres permettent de définir tous les ordonnancements statiques, et ils permettent de synthétiser la formulation des divers problèmes. Détaillons ces trois paramètres :

- Le paramètre $\boldsymbol{\alpha}$ définit le type de processeurs et leur nombre :
  ⋆ $\boldsymbol{\alpha \in \{P, \bar{P}, P_m\}}$
  – Si $\alpha = P$, ça signifie qu'on a des processeurs identiques et que leur nombre m est une entrée du problème.
  – Si $\alpha = \bar{P}$ ou $\alpha = \infty$, ça signifie qu'on a des processeurs identiques, leur nombre étant suffisant ou non borné.
  – Si $\alpha = P_m$, ça signifie qu'on a un nombre de processeurs identiques fixé.

- Le paramètre $\boldsymbol{\beta}$ définit le type des tâches à exécuter, leur temps d'exécution, il décrit aussi les différentes contraintes à respecter comme les précédences, la possibilité de préemption, les dates limites, la périodicité. De manière synthétique, on obtient $\boldsymbol{\beta = \beta_1\beta_2\beta_3\beta_4\beta_5}$ où chaque β_i décrit un de ces points.
  ⋆ $\boldsymbol{\beta_1}$ indique la structure d'un graphe de précédence s'il existe.
  ⋆ $\boldsymbol{\beta_2}$ spécifie les communications entre processeurs.
  ⋆ $\boldsymbol{\beta_3}$ donne les caractéristiques pour les tâches présentes.
  ⋆ $\boldsymbol{\beta_4}$ symbolise la présence de date limite d'exécution pour les tâches.
  ⋆ $\boldsymbol{\beta_5}$ et les $\boldsymbol{\beta_i}$ suivants sont utilisés pour décrire d'autres caractéristiques inhabituelles.

- Le paramètre $\boldsymbol{\gamma}$ définit le type d'objectifs à optimiser, il peut y en avoir plusieurs mais dans ce manuscrit, nous n'étudierons qu'un seul objectif à la fois.

Définition 2.1.12 (Date de début d'exécution) :
On note $\boldsymbol{\sigma(i)}$ la date de début d'exécution de la tâche i pour un ordonnancement donné.

Définition 2.1.13 (Date de complétion) :
On note $\boldsymbol{C(i)}$ *la date de fin d'exécution de la tâche i pour un ordonnancement donné.*

Définition 2.1.14 (Le date de complétion maximum «Makespan») :
Le principal objectif que nous étudierons sera le temps de complétion maximum, noté $\boldsymbol{C_{max}}$*, qui est défini comme étant le temps de complétion maximum parmi toutes les tâches. En littérature il est souvent nommé «**makespan**», et ainsi pour un nombre n de tâches à exécuter* $C_{max} = max\{C(1), C(2), \ldots, C(n)\}$.

Définition 2.1.15 (La somme minimum des dates de complétion) :
Le second objectif que nous étudierons sera la somme minimum des dates de complétion, noté $\sum_{\boldsymbol{i}} \boldsymbol{C(i)}$*, qui est défini comme étant le* $min\{C(1) + C(2) + \ldots + C(n)\}$ *pour un nombre n de tâches à exécuter. Il s'agit donc ici de minimiser la moyenne de la fin d'exécution des tâches.*

Définition 2.1.16 ($\boldsymbol{T_{seq}}$ et $\boldsymbol{T_{idle}}$) :
Nous notons $\boldsymbol{T_{seq}}$ *la somme des temps d'exécution de toutes les tâches à exécuter. Et* $\boldsymbol{T_{idle}}$ *la somme des temps d'inactivité du processeur entre le début d'exécution de la première tâche ordonnancée et la fin d'exécution de la dernière.*

2.1.4 Éléments de la théorie de la complexité

La théorie de la complexité cherche à connaître formellement la difficulté de répondre à un problème algorithmique. un problème algorithmique est un problème énoncé de façon mathématique avec des hypothèses, des données et une question. Il y a deux types de problèmes :

- Problème de **décision** : la réponse est oui ou non. Le problème est dit **décidable** si sa réponse peut être fournie par un algorithme.
- Problème d'**existence** : la réponse consiste à fournir un élément en particulier. Le problème est **calculable** si l'élément calculé peut être fourni par un algorithme.

Parmi les problèmes d'existence, nous étudierons exclusivement les problèmes d'**optimisation** qui consistent à trouver une des meilleures solutions dans un ensemble des solutions réalisables. Cet ensemble peut être fini mais compter un très grand nombre d'éléments. La qualité d'une solution s'évalue par une **fonction objective**, la meilleure solution est celle dont la qualité correspond le plus à un objectif recherché (généralement maximiser ou minimiser une quantité). Par la suite nous parlerons soit de problème de décision, soit d'optimisation.

La donnée d'un problème de décision ou d'optimisation est appelée une **instance** du problème. Nous utiliserons ce terme par la suite.

Notons qu'un problème d'existence peut être transformé en un problème de décision équivalent. La théorie de la complexité vise à savoir si la réponse à un problème peut être donnée efficacement ou au contraire être inatteignable en pratique (et en théorie) en se basant sur une estimation théorique des temps de calcul et des besoins en mémoire informatique.

Machines déterministes

L'un des modèles de calcul les plus utilisés est celui des machines abstraites dans la lignée du modèle proposé par Alan Turing en 1936 [68]. Ces modèles supposent un ordinateur idéal avec une mémoire infinie, dans lequel une cellule mémoire est accessible en un temps constant, et les instructions sont exécutées séquentiellement.

Les **machines déterministes** sont telles que chaque action possible est unique, c'est-à-dire que l'action à effectuer est dictée de façon unique par l'état courant de celle-ci. S'il peut y avoir plusieurs choix possibles d'actions à effectuer, la machine est dite non déterministe. Le lecteur est renvoyé à [68, 69] pour une description détaillée de ce modèle de machines. Pour cette architecture, le codage d'une donnée est supposé raisonnable. Par exemple :

- Le codage d'un nombre $\boldsymbol{n}$ utilise $\boldsymbol{log(n)}$ bits.
- Le codage d'un graphe[1] $\boldsymbol{G = (V, E)}$ est polynomial en $|\boldsymbol{V}|$

Dans le but de mieux regrouper les problèmes les uns par rapport aux autres, la théorie de la complexité établit des hiérarchies de difficulté entre les problèmes algorithmiques selon des classes de complexité.

Les classes de complexité

Nous présentons quelques classes de complexité dans lesquelles les problèmes de décision et d'existence se situent. Pour plus d'informations sur la théorie de la complexité, nous renvoyons le lecteur vers les ouvrages de Garey et Johnson [31], et de Paschos [58].

Définition 2.1.17 (Classe $\mathcal{P}$) :
Un problème est dans $\boldsymbol{\mathcal{P}}$ s'il peut être résolu par un algorithme déterministe en temps polynomial par rapport à la taille de l'instance. On qualifie alors le problème de ***polynomial****, c'est un problème de complexité $O(n^k)$ pour un certain k.*

Définition 2.1.18 (Classe $\mathcal{NP}$) :
La classe $\boldsymbol{\mathcal{NP}}$ des problèmes ***Non-déterministes Polynomiaux*** *réunit les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. De façon équivalente, c'est la classe des problèmes pour lesquels il existe un algorithme polynomial déterministe $\mathcal{A}$ et, pour chaque instance positive $\boldsymbol{I}$ un* ***certificat cert(I)****, tels que l'algorithme $\mathcal{A}$ sait reconnaître à l'aide de $\boldsymbol{cert(I)}$ que la réponse est oui pour $\boldsymbol{I}$. Intuitivement, ce sont les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial. Le nombre de solutions peut néanmoins être exponentiellement grand.*

On a l'inclusion $\boldsymbol{\mathcal{P} \subseteq \mathcal{NP}}$, mais on ne sait pas si cette inclusion est stricte, ou si $\boldsymbol{\mathcal{P} = \mathcal{NP}}$. Une grande part de la théorie de la complexité s'est construite sous l'hypothèse que $\boldsymbol{\mathcal{P} \neq \mathcal{NP}}$.

Définition 2.1.19 (Classe $\mathcal{NPO}$) :
La classe $\boldsymbol{\mathcal{NPO}}$ est formée des problèmes d'optimisation qui vérifient les propriétés suivantes :

1. *La faisabilité d'une solution est vérifiable en temps polynomial.*
2. *La valeur d'une solution réalisable est calculable en temps polynomial.*
3. *Une solution réalisable est calculable en temps polynomial.*

Proposition 2.1.1 :
La valeur décisionnelle d'un problème d'optimisation dans $\boldsymbol{\mathcal{NPO}}$ appartient à $\boldsymbol{\mathcal{NP}}$.

Définition 2.1.20 (Classe $\mathcal{NP} - complet$) :
La classe $\boldsymbol{\mathcal{NP} - complet}$ est une sous-classe de $\boldsymbol{\mathcal{NP}}$ constituée des problèmes de décision qui sont les plus difficiles de $\boldsymbol{\mathcal{NP}}$, et de difficulté équivalente par rapport à leur résolution polynomiale. Plus formellement, un problème de décision Π est $\boldsymbol{\mathcal{NP} - complet}$ si et seulement s'il vérifie ces deux conditions :

[1] Le codage d'un graphe utilise de l'ordre de $|\boldsymbol{V}| + |\boldsymbol{E}|$ cases de mémoires. Une case de mémoire est un emplacement pouvant stocker un entier.

1. $\Pi \in \boldsymbol{\mathcal{NP}}$,
2. $\forall\ \Pi' \in \boldsymbol{\mathcal{NP}}$, Π' *se réduit à* Π *par un algorithme polynomial.*

Un problème qui vérifie la condition 2 (sans pour autant la condition 1) est appelé $\boldsymbol{\mathcal{NP}}$-difficile ($\boldsymbol{\mathcal{NP}}$-hard dans la littérature anglaise). On peut donc dire qu'un problème est $\boldsymbol{\mathcal{NP}}$-complet si et seulement s'il appartient à $\boldsymbol{\mathcal{NP}}$ et est $\boldsymbol{\mathcal{NP}}$-difficile.

Ainsi, pour démontrer qu'un problème Π est $\boldsymbol{\mathcal{NP}}$-complet, il suffit de démontrer les deux propriétés suivantes :
- $\Pi \in \boldsymbol{\mathcal{NP}}$,
- Un problème quelconque Π' $\boldsymbol{\mathcal{NP}}$-complet se réduit à Π par un algorithme polynomial.

Pour finir avec les classes de complexité que nous étudierons, notons que la notion de $\boldsymbol{\mathcal{NP}}$-complétude est propre aux problèmes de décision. Lorsque nous traiterons des problèmes d'optimisation, le terme approprié utilisé dans la littérature francophone, est $\boldsymbol{\mathcal{NP}}$-difficile[2]. Nous avons alors la définition suivante :

Définition 2.1.21 (Problème d'optimisation $\boldsymbol{\mathcal{NP}}$-difficile) :
Un problème de $\boldsymbol{\mathcal{NPO}}$ *est* $\boldsymbol{\mathcal{NP}}$*-difficile si et seulement si sa variante décisionnelle est un problème* $\boldsymbol{\mathcal{NP}}$*-complet.*

Réductions polynomiales

En 1971, S. Cook démontre que le problème **SAT** est $\boldsymbol{\mathcal{NP}}$-complet [23], il classe ainsi le premier problème $\boldsymbol{\mathcal{NP}}$-complet. À partir de ce problème, la Définition 2.1.20 a permis de classifier les problèmes $\boldsymbol{\mathcal{NP}}$-complets de la littérature grâce à des réductions polynomiales.

Le principe de réduction d'un problème Π à un problème Π' consiste à étudier à travers une transformation le problème Π comme un cas particulier de Π'. Si on sait résoudre le problème Π' et que la transformation se fait en temps polynomial, alors on sait résoudre le problème Π en temps polynomial. La réduction est un outil puissant, servant à classifier la difficulté des problèmes, ou encore à résoudre des problèmes. La réduction classique utilisé pour les problèmes de décision en théorie de la complexité est la réduction de Turing.

Définition 2.1.22 (Réduction de Turing) :
Soient deux problèmes de décision Π *et* Π'*, une* ***réduction de Turing****, noté* $\leq_T$ *dans la suite, est une fonction* $f : \mathcal{I}_\Pi \longrightarrow \mathcal{I}_{\Pi'}$ *qui satisfait ces deux conditions :*

1. f *est calculable en temps polynomial,*
2. *Pour toute instance de* $I_1 \in \mathcal{I}_\Pi$*,* I *donne une réponse positive au problème de décision* Π *si et seulement si* $f(I_1)$ *donne également une réponse positive au problème de décision* Π'*.*

Proposition 2.1.2 :
Les rédutions de Turing sont transitives, si $\Pi_1 \leq_T \Pi_2$ *et* $\Pi_2 \leq_T \Pi_3$*, alors* $\Pi_1 \leq_T \Pi_3$*.*

Lemme 2.1.1 :
Si $\Pi \leq_T \Pi'$*, alors* $\Pi' \in \mathcal{P}$ *implique* $\Pi \in \mathcal{P}$ *(et inversement,* $\Pi \notin \mathcal{P}$ *implique* $\Pi' \notin \mathcal{P}$*).*

Et nous pouvons donner un dernier lemme sur la preuve de $\boldsymbol{\mathcal{NP}}$-complétude d'un problème de décision :

[2] À ne pas confondre avec le même terme lorsqu'il est utilisé au sens de la propriété 2 de la Définition 2.1.20. Comme le dit V. Th. Paschos dans [58], aucun des ouvrages classiques n'explique le pourquoi de cette collision.

Lemme 2.1.2 :
On peut montrer qu'un problème Π *est* $\boldsymbol{\mathcal{NP}}$*-complet en démontrant les points suivants :*

1. $\Pi \in \boldsymbol{\mathcal{NP}}$,
2. $\exists\ \Pi'$ $\boldsymbol{\mathcal{NP}}$*-complet tel que* $\Pi' \leq_T \Pi$.

Pour finir cette section sur l'étude de la complexité de certains problèmes, nous donnons un lemme sur l'implication d'une hiérarchie de difficulté entre deux problèmes permettant d'obtenir la complexité d'un des deux problèmes selon la complexité de l'autre.

Lemme 2.1.3 :
Soient deux problèmes $\mathcal{NP}$ Π_1 *et* Π_2 *tels que* Π_1 *est une généralisation de* Π_2 *(i.e.* Π_2 *est une instance particulière de* Π_1*). Si* Π_2 *est* $\mathcal{NP}$*-complet, alors forcément* Π_1 *l'est aussi. Et inversement si* Π_1 *est polynomial, alors forcément* Π_2 *l'est aussi.*

2.1.5 Éléments sur les approximations en ordonnancement

Les problèmes d'ordonnancement concrets ont généralement une structure trop complexe pour que l'on puisse espérer obtenir une garantie théorique satisfaisante sur la qualité des solutions des heuristiques. Dans ce cas, il faut procéder à des jeux d'essais pour estimer une garantie expérimentale. Cependant, nous n'avons pas retenu ce choix pour juger de la qualité d'un algorithme traitant des applications générales.

Nous avons choisi de démontrer une garantie théorique de manière déterministe (ce choix est parfois possible pour des problèmes d'ordonnancement de base). Pour évaluer la performance relative d'une heuristique nous utiliserons la définition suivante :

Définition 2.1.23 (Ratio de performance ou rapport d'approximation) :
Soit h *un algorithme d'ordonnancement. Le ratio de performance de* h*, noté* $\boldsymbol{\rho(h)}$*, est le maximum sur toutes les instances entre la durée donnée par un ordonnancement fourni par* h *et la durée de l'ordonnancement optimal, c'est à dire :*

$$\boldsymbol{\rho(h) = max_G \frac{C^h_{max}}{C^{opt}_{max}}(G)}$$

On dit qu'un algorithme h *est* ρ*-approché (ou encore* ρ*-compétitif) si* $\rho(h) \leq \rho$.
Un algorithme h *est d'autant plus efficace que son ratio* ρ *est petit. La démarche dans le cas d'un problème* $\mathcal{NP}$*-complet est donc de trouver le meilleur algorithme possible. En fait la plupart du temps nous ne sommes capables que de donner une borne supérieure du ratio du meilleur algorithme (en étudiant le ratio d'un algorithme connu) et une borne inférieure de ce ratio.*
Par la suite nous utiliserons l'appellation heuristique pour parler d'un algorithme polynomial appliqué à un problème $\mathcal{NP}$*-difficile (donc qui ne donne qu'une approximation sans garantie de performance).*

Approximation dépendant de l'instance

Les rapports dépendant de l'instance sont souvent des fonctions soit de la taille même de l'instance d'un problème, soit d'autres paramètres de cette instance :

- **APX**, la classe des problèmes $\mathcal{NPO}$ approximables à facteur constant, si et seulement si il existe un algorithme polynomial approché A pour un problème Π et une constante fixée $r \in \mathbb{R}^+$, tels que le rapport d'approximation de A est borné par r.
- **NON-APX**, la classe des problèmes $\mathcal{NPO}$ non approximables à facteur constant si et seulement si il n'existe aucun algorithme avec garantie de performance polynomial approché A pour un problème Π tel que le rapport d'approximation est borné par une constante.

2.2 Liste des problèmes utilisés

Cette section regroupe la liste des problèmes utilisés tout au long du manuscrit. Nous les présentons en deux parties, les problèmes appartenant à $\mathcal{P}$ et ceux qui sont $\mathcal{NP}$-complets.

2.2.1 Les problèmes polynomiaux

Données	:	Soit $G = (V, E)$ un graphe.
Résultat	:	Un couplage M dans G.
Objectif	:	M est de taille maximum, c'est à dire qu'il contient le plus grand nombre d'arêtes possible.

Problème d'optimisation 2.2.1: COUPLAGE MAXIMUM

Données	:	Soit $G = (V, E)$ un graphe, une fonction de poids Ω pour chaque arête e_i, où $\|V\| = n \equiv 0(2)$.
Résultat	:	Un couplage parfait $M = \{e_1, e_2, \ldots, e_{\frac{n}{2}}\}$.
Objectif	:	Trouver M tel que $\sum_{e_i \in M} \Omega(e_i)$ soit le plus grand possible.

Problème d'optimisation 2.2.2: COUPLAGE PARFAIT DE POIDS MAXIMUM

Données	:	Soit $G = (V, E)$ un graphe.
Résultat	:	Une clique K, c'est à dire un sous-graphe complet de G induit par K sommets de G.
Objectif	:	K est de taille maximal, c'est à dire qu'il n'existe pas dans G une clique contenant K.

Problème d'optimisation 2.2.3: CLIQUE MAXIMALE

2.2.2 Les problèmes $\mathcal{NP}$-complets

Un grand nombre de problèmes de la théorie des graphes est $\mathcal{NP}$-complet, et donc si $\mathcal{NP} \neq \mathcal{P}$ ces problèmes sont connus pour ne pas avoir d'algorithme en temps polynomial permettant de les résoudre. **Garey et Johnson** [31], **Ausiello et al.** [4] proposent une description de la théorie de la $\mathcal{NP}$-complétude, une liste de problèmes $\mathcal{NP}$-complets connus, et une bibliographie sur le sujet. En particulier, ils listent plusieurs problèmes sur le recouvrement de sommets $\mathcal{NP}$-complets incluant le recouvrement en cliques et le recouvrement par des sous-graphes isomorphes.

La plupart des problèmes présentés ci-dessous sont répertoriés dans [31] à l'aide d'une référence comme par exemple [$GT11$] pour le problème PARTITION EN TRIANGLES.

Données	:	Soit $G = (V, E)$ un graphe, avec $\|V\| = 3q$ pour un certain entier q.
Question	:	Les sommets de G peuvent ils être partitionnés en q ensembles disjoints $V_1, V_2, \ldots, V_q$, chacun contenant exactement trois sommets, et tel que pour chaque $V_i = \{u_i, v_i, w_i\}$, $1 \leq i \leq q$, les trois arêtes $\{u_i, v_i\}$, $\{u_i, w_i\}$, et $\{v_i, w_i\}$ appartiennent à E ?

Problème de décision 2.2.1: PARTITION EN TRIANGLES [$GT11$]

Données : Soit $G = (V, E)$ un graphe, et un entier positif $K \leq |V|$.
Question : Est-ce que G contient une clique de taille K ou plus, c'est à dire un sous-ensemble $V' \subseteq V$ avec $|V'| \geq K$ tel que tous les sommets de V' sont reliés deux à deux par une arête ?

Problème de décision 2.2.2: CLIQUE [$GT19$]

Données : Soit $G = (V, E)$ un graphe, et un entier positif $K \leq |V|$.
Question : Les sommets de G peuvent ils être partitionnés en $k \leq K$ ensembles disjoints $V_1, V_2, \ldots, V_k$ tels que, pour $1 \leq i \leq k$, le sous-graphe induit par V_i est un graphe complet ?

Problème de décision 2.2.3: PARTITION EN CLIQUES [$GT15$]

Données : Soient $G = (V, E)$ et $H = (V_H, E_H)$ deux graphes avec $|V| = q|V_H|$ pour n'importe quel $q \in \mathbb{Z}^+$.
Question : Les sommets de G peuvent ils être partitionnés en q ensembles disjoints $V_1, V_2, \ldots, V_q$ tels que, pour $1 \leq i \leq q$, le sous-graphe de G induit par V_i est isomorphe à H ?

Problème de décision 2.2.4: PARTITION EN SOUS-GRAPHES ISOMORPHES [$GT15$]

Données : Soit $G = (V, E)$ un graphe.
Question : Est-ce que G contient une chaîne passant une seule fois par tous les sommets de V sans jamais utiliser deux fois une même arête de E ?

Problème de décision 2.2.5: CHAÎNE HAMILTONIENNE [$GT39$]

Données : Soit $G = (V, E)$ un graphe, et un entier positif $K \leq |V|$.
Question : Les sommets de G peuvent ils être partitionnés en $k \leq K$ ensembles disjoints $V_1, V_2, \ldots, V_k$ tels que, pour $1 \leq i \leq k$, le sous-graphe induit par V_i contient une chaîne ?

Problème de décision 2.2.6: PARTITION EN CHAÎNES [$GT39$]

Données : Soit $G = (V, E)$ un graphe, et deux entiers positifs K et $P \leq |V|$.
Question : Les sommets de G peuvent ils être partitionnés en $p \leq P$ ensembles disjoints $V_1, V_2, \ldots, V_p$ tels que, pour $1 \leq i \leq p$, le sous-graphe induit par V_i contient une chaîne de longueur inférieure ou égale à K ?

Problème de décision 2.2.7: PARTITION EN K-CHAÎNES (*Steiner*[67])

Données	:	Soit Q un ensemble fini d'éléments, et une taille $s(e) \in \mathbb{Z}^+$ pour chaque $e \in Q$.
Question	:	Est-ce que Q peut être partitionné en deux ensembles disjoints Q_1 et Q_2 tels que $\sum_{e \in Q_1} s(e) = \sum_{e \in Q_2} s(e) = B$?

Problème de décision 2.2.8: PARTITION [$SP12$]

Données	:	Soit Q un ensemble de $3q$ éléments $Q = \{e_1, e_2, \ldots, e_{3q}\}$, et une borne $E \in \mathbb{Z}^+$, tels que $\forall i \ \frac{E}{4} < e_i < \frac{E}{2}$ et $\sum_{e_i \in Q} e_i = qE$.
Question	:	Est-ce que Q peut être partitionné en q ensembles disjoints $Q_1, Q_2, \ldots, Q_q$ tels que, pour $1 \leq i \leq q$, $\sum_{e_i \in Q_j} e_i = E$ (notons que chaque Q_i doit alors contenir exactement trois éléments de Q) ?

Problème de décision 2.2.9: 3-PARTITION [$SP15$]

2.3 Outils mathématiques

Dans cette section, nous allons définir un type de recouvrement de sommets d'un graphe ne comprenant que des arêtes et des chaînes de longueur 2. Cette méthode sera utilisée dans la Section 6.3.2 à la page 95.

2.3.1 Le 2-recouvrement : extension de la notion de couplage

Pour commencer, nous allons introduire et définir le 2-recouvrement.

Définition 2.3.1 (2-recouvrement) :
Un ***2-recouvrement M****, dans un graphe non orienté $G = (V, E)$, est un ensemble d'arêtes tel que les composantes connexes du graphe partiel induits par les arêtes de M sont soit des sommets isolés, soit des* ***arêtes****, soit des* ***chaînes de longueur 2****.*

Définition 2.3.2 (Sommet M-saturé) :
Nous appelons un ***sommet M-saturé*** *(respectivement* ***M-non-saturé****) relativement à M un sommet qui appartient (respectivement n'appartient pas) à une arête de M. L'ensemble des sommets M-saturés (respectivement M-non-saturés) sera noté* $\boldsymbol{S(M)}$ *(respectivement* $\boldsymbol{NS(M)}$*).*

Définition 2.3.3 (Le degré d'un sommet relativement à M) :
Soit M un 2-recouvrement dans un graphe $G = (V, E)$. Pour chaque $i = 1, \ldots, k$, soit $\boldsymbol{d_M(x_i)}$ *le nombre d'arêtes de M qui sont incidentes à x_i.*

Définition 2.3.4 (2-recouvrement maximum) :
Un ***2-recouvrement maximum*** *est un 2-recouvrement où le nombre de sommets saturés est maximum, et par conséquent le nombre de sommets isolés est minimum.*

Définition 2.3.5 (2-recouvrement parfait) :
Un ***2-recouvrement parfait*** *est un 2-recouvrement M tel que chaque sommet de G est saturé, c'est à dire* $\boldsymbol{S(M) = |V|}$.

Nous allons maintenant nous intéresser à la définition d'une chaîne alternée dans un 2-recouvrement qui est analogue à celle définie dans le cadre de l'étude d'un couplage de taille maximum [8].

Définition 2.3.6 (Chaîne M-alternée) :
*Soit M un 2-recouvrement dans un graphe $G=(V,E)$, une chaîne **M-alternée** $C=x_0,x_1,\ldots,x_k$ est une chaîne dans G telle que pour $i=0,\ldots,\lceil\frac{k}{2}\rceil-1$:*

- $x_0 \in NS(M)$
- $\{x_{2i},x_{2i+1}\} \notin M$
- *si* $k \neq (2i+1)$, $\{x_{2i+1},x_{2i+2}\} \in M$

Définition 2.3.7 (Colonne vertébrale associée à une chaîne M-alternée) :
*Soit M un 2-recouvrement dans un graphe $G=(V,E)$, et $C=x_0,x_1,\ldots,x_k$ une chaîne M-alternée dans G. Une **colonne vertébrale**, noté T, associé à C est composé de l'union de la chaîne M-alternée C, des sommets reliés à la chaîne C par une arête appartenant à M, et éventuellement des extrémités de ces sommets (voir Figure 2.3).*

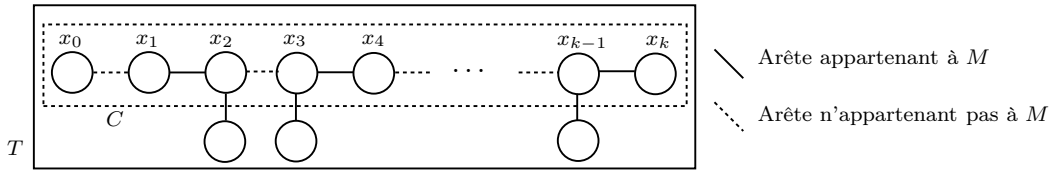
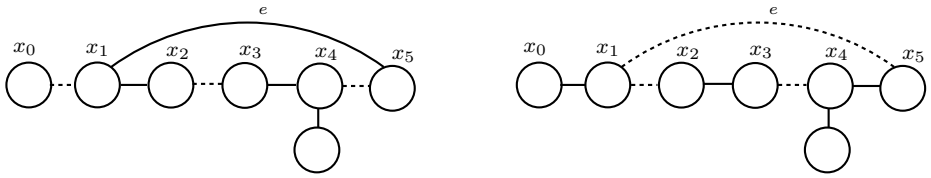


Fig. 2.3 – Exemple d'une colonne vertébrale T associée à une chaîne M-alterné C

Remarque 2.3.1 :
De plus, notons que si T contient un cycle, il existe une arête $e \in M$ qui relie deux sommets non-adjacents de C. Si un de ces sommets n'est pas une extrémité de C, alors nous avons une chaîne de longueur trois dans M (tous les sommets de C sont couverts par des arêtes de M à l'exception éventuellement des extrémités). Par définition, $x_0 \in NS(M)$, ainsi T contient un cycle lorsque le dernier sommet x_k de C est connecté à un autre sommet de C par une arête $e \in M$ et $e \notin C$ (voir l'illustration de la Figure 2.4(a)). Notons que $d_M(x_k)=1$.



(a) Exemple d'un cycle dans T

(b) Exemple d'une augmentation en présence d'un cycle dans T

Fig. 2.4 – Illustrations pour les Remarques 2.3.1 et 2.3.2

Définition 2.3.8 (Chaîne M-alternée augmentante) :
*Soit $C=x_0,x_1,\ldots,x_k$ une chaîne M-alternée, avec $x_0 \in NS(M)$. C est une **chaîne M-alternée augmentante** si la cardinalité du 2-recouvrement augmente (c'est à dire que le nombre de sommets non-saturés est réduit d'au moins un) en changeant éventuellement l'appartenance à M des arêtes de C.*

Remarque 2.3.2 :
D'après la Remarque 2.3.1, une chaîne de longueur trois ou quatre peut être créée par l'opération

d'augmentation utilisée dans la Définition 2.3.8. Soit e une arête de M qui crée le cycle dans T (créant ainsi la chaîne de longueur trois ou quatre après l'opération d'augmentation). Alors l'arête e peut être retirée de M dans le but d'accroître le nombre de sommets couverts par un 2-recouvrement, tout en respectant la définition d'un 2-recouvrement (voir l'illustration de la Figure 2.4(b)).

Cette remarque entraîne la propriété suivante sur les cycles :

Proposition 2.3.1 :
Soit $C = x_0, x_1, \ldots, x_k$ une chaîne M-alternée, avec $x_0 \in NS(M)$, et T la colonne vertébrale associée à C. Si T contient un cycle alors C est une chaîne M-alternée augmentante.

Définition 2.3.9 (Sommet pair et impair) :
Soit $C = x_0, x_1, \ldots, x_k$ une chaîne M-alternée, avec $x_0 \in NS(M)$. Un sommet x_i avec un indice égale à un nombre pair (resp. nombre impair) dans C est appelé un ***sommet pair*** *(resp.* ***sommet impair****).*

Nous allons maintenant donner un lemme portant sur les chaînes M-alternés augmentantes, et un théorème fondamental pour le 2-recouvrement.

Lemme 2.3.1 :
Soit M un 2-recouvrement, $C = x_0, x_1, \ldots, x_k$ une chaîne M-alternée, avec $x_0 \in NS(M)$, et T la colonne vertébrale associée à C. C est une chaîne M-alternée augmentante si et seulement si il existe un sommet impair x_{2i-1}, $i \in \mathbb{N}^*$, tel que $d_M(x_{2i-1}) \neq 2$.

Preuve par contradiction :

$\Rightarrow$ Supposons que C soit une chaîne M-alternée augmentante, et supposons qu'un sommet impair x_{2i-1} tel que $d_M(x_{2i-1}) \neq 2$ n'existe pas (donc T ne contient aucun cycle). Ainsi, C et sa colonne vertébrale associée T sont de la forme de la Figure 2.5.

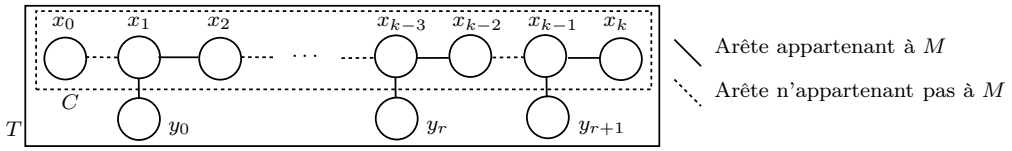


Fig. 2.5 – Squelette de la colonne vertébrale T associée à C

D'après la Définition 2.3.8 et la Remarque 2.3.2, si T ne contient aucun cycle, nous pouvons simplement augmenter le nombre de sommets saturés dans la chaîne C en changeant l'appartenance à M des arêtes de C. Si nous changeons l'appartenance à M de l'arête $\{x_0, x_1\}$ dans le but de saturer x_0, l'appartenance de l'arête $\{x_1, x_2\}$ doit changer, sinon x_1 serait le centre d'une **étoile**[3]. Pour cela, nous changeons l'appartenance à M de l'arête $\{x_1, x_2\}$, qui implique que nous devons changer $\{x_2, x_3\}$. Récursivement, nous changeons l'appartenance à M de toutes les arêtes de C. Ainsi, le dernier sommet x_k ne sera pas saturé, et C ne sera pas une chaîne M-alternée augmentante. Ce qui est en contradiction avec les hypothèses de départ. Donc il existe un sommet x_{2i-1} tel que $d_M(x_{2i-1}) \neq 2$.

$\Leftarrow$ Si T contient un cycle, alors C est une chaîne M-alternée augmentante d'après la Propriété 2.3.1. Sinon, supposons que T ne contient pas de cycle, mais qu'il existe un sommet impair

[3]Le centre d'une étoile est le seul sommet de l'étoile ayant un degré plus grand que un.

x_{2i-1}, $i \in \mathbb{N}^*$, tel que $d_M(x_{2i-1}) \neq 2$. Nous allons montrer que C est une chaîne M-alternée augmentante. Soit $x_j = x_{2i-1}$, $i \in \mathbb{N}^*$, le premier sommet impair sur la chaîne M-alternée tel que $d_M(x_j) < 2$. Nous avons trois cas possibles :

1. Supposons que $\boldsymbol{d_M(x_j) = 0}$, la chaîne M-alternée C se termine avec un sommet non-saturé. Ainsi, C est une chaîne M-alternée augmentante (voir l'illustration de la Figure 2.6.a).
2. Supposons que $\boldsymbol{d_M(x_j) = 1}$ et $\boldsymbol{d_M(x_{j+1}) = 1}$, la chaîne M-alternée C contient une arête $\{x_j, x_{j+1}\} \in M$ dont les extrémités ont un degré égal à 1. Nous retirons la partie de la chaîne qui se trouve après cette arête, puisque cette partie est déjà couverte. Ainsi, nous avons une sous-chaîne M-alternée, dans laquelle tous les sommets d'indice impair ont un degré égal à 2, et où la fin de la sous-chaîne est une arête $\{x_j, x_{j+1}\}$. Il est facile de voir que cette sous-chaîne est une chaîne M-alternée augmentante où il suffit de changer l'appartenance à M des arêtes de C, à l'exception de la dernière $\{x_j, x_{j+1}\}$. Donc C est une chaîne M-alternée augmentante (voir l'illustration de la Figure 2.6.b).
3. Supposons que $\boldsymbol{d_M(x_j) = 1}$ et $\boldsymbol{d_M(x_{j+1}) = 2}$, la chaîne M-alternée C possède un sommet impair de degré égal à 1, adjacent à un sommet pair de degré égal à 2. Nous retirons la partie de la chaîne qui se trouve après ce sommet pair de degré égal à 2, cette partie est déjà couverte. Ainsi, nous avons une sous-chaîne M-alternée dans laquelle tous les sommets d'indice impair ont un degré égal à 2, et où la fin de la sous-chaîne est une chaîne de longueur 2. Il est facile de voir que cette sous-chaîne est une chaîne M-alternée augmentante où il suffit de changer l'appartenance à M des arêtes de C. Et donc, C est une chaîne M-alternée augmentante (voir l'illustration de la Figure 2.6.c).

□

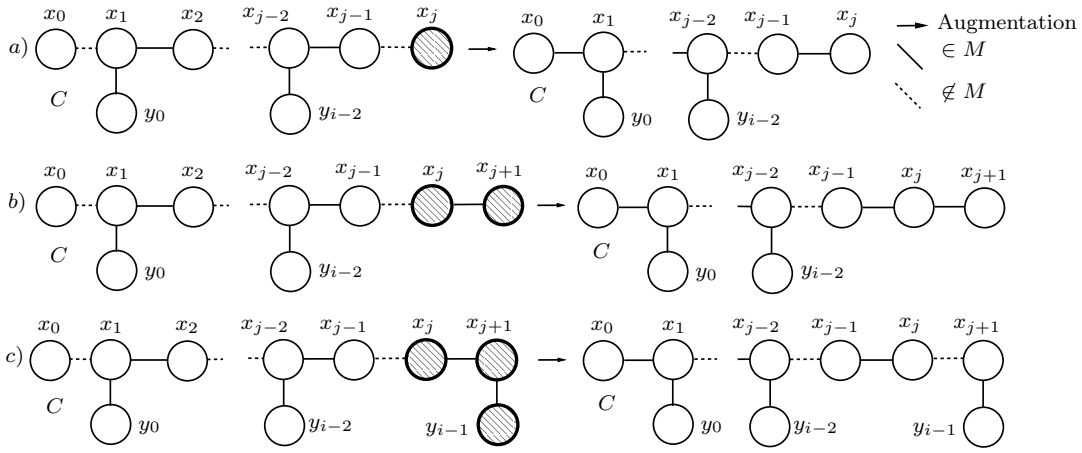


Fig. 2.6 – Opération d'augmentation dans les trois cas avec $j = 2i - 1$

Avant de donner le théorème fondamental, nous définissons deux types de sommets dans une chaîne M-alternée non-augmentante afin de faciliter la preuve du théorème qui suit :

Définition 2.3.10 (Feuille et racine) :
*Soit M un 2-recouvrement dans un graphe G, et soit C une chaîne M-alternée non-augmentante. Une feuille (resp. une racine) est défini comme étant un sommet qui admet seulement un voisin (resp. deux voisins) dans M. Tout sommet $x_j \in C$ avec $j = 2i$ est une **feuille**, et tous les sommets $y_i \in T \backslash C$ sont aussi des feuilles, $i \in \mathbb{N}$. Inversement, tout sommet $x_j \in C$ avec $j = 2i + 1$ est une **racine**, $i \in \mathbb{N}$.*

Théorème 2.3.1 :
Un 2-recouvrement M d'un graphe G est maximum si et seulement si G ne possède pas de chaîne M-alternée augmentante.

Preuve :

Cette preuve est basée sur la preuve classique du couplage maximum donné par [8].

$\Rightarrow$ Soit M un 2-recouvrement maximum de G. Si G admet une chaîne M-alternée augmentante, la cardinalité de M doit être augmentée d'après le Lemme 2.3.1, ce qui aboutirait à une contradiction car M ne serait pas maximum.

$\Leftarrow$ Soit M_1 un 2-recouvrement dans G. Supposons que G ne possède pas de chaîne M_1-alternée augmentante.

Raisonnant par l'absurde, supposons que M_1 ne soit pas maximum, et soit M_2 un autre 2-recouvrement dans G qui est maximum. Clairement, M_2 couvre plus de sommets que M_1. À partir de ces hypothèses, la structure suivante est définie (voir Figure 2.7) :

- Supposons que M_2 couvre K sommets non-saturés par M_1, cet ensemble est noté $\boldsymbol{S_1 = \{x_i | x_i \in S(M_2) \cap NS(M_1)\}}$.
- À partir de n'importe quel sommet x_i de S_1, il existe nécessairement une arête dans G reliant x_i à une chaîne M_1-alternée non-augmentante. Soit S_2 l'ensemble des sommets saturés par M_1, et qui appartiennent à ces chaînes M_1-alternées non-augmentantes. $\boldsymbol{|S_2| = 3N}$ est le nombre de sommets dans ces chaînes avec N racines et $2N$ feuilles.
- D'après les hypothèses, nous savons qu'il peut exister des sommets saturés par M_1 qui n'appartiennent pas à l'ensemble S_2. Ces sommets sont saturés soit par des arêtes soit par des chaînes de longueur 2. Soit $\boldsymbol{S_3 = \{x_i | x_i \in S(M_1) \wedge x_i \notin S_2\}}$ l'ensemble de ces sommets.
- Enfin, il existe des sommets non-saturés ni par M_1 ni par M_2. Cet ensemble est noté $\boldsymbol{S_4 = \{x_i | x_i \in NS(M_1) \cap NS(M_2)\}}$.

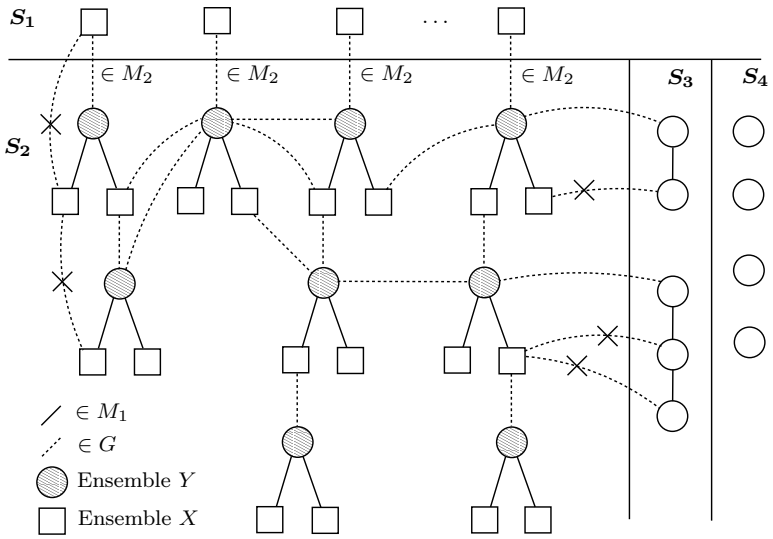


Fig. 2.7 – Schéma de la preuve du Théorème 2.3.1

À partir de cette structure, nous pouvons donner les propriétés suivantes :

1. Les sommets de S_1 forment nécessairement un **stable**. En effet, si ce n'est pas le cas il existerait deux sommets non-saturés par M_1 reliés par une arête, et nous aurions une chaîne M_1-alternée augmentante.

2. Dans l'ensemble S_2, il peut exister des arêtes de G reliant une racine à n'importe quel sommet de S_1 et S_2. De la même manière, une feuille de S_2 ne peut être reliée ni à une feuille de S_2 ni à un sommet de S_1, par une arête de G. En effet, dans les deux cas nous aurions soit un cycle soit une chaîne M_1-alternée augmentante (voir l'illustration de la Figure 2.7).
3. Chaque racine de S_2 peut être reliée par une arête de G à n'importe quel sommet de S_3. Si une feuille de S_2 est reliée par une arête de G à l'extrémité d'une arête d'une chaîne de longueur 2 de S_3, alors il existerait une chaîne M_1-alternée augmentante. Et si une feuille de S_2 est reliée au centre d'une chaîne de longueur 2 de S_3, cette chaîne appartiendrait à S_2 par définition. Donc en conclusion, les feuilles de S_2 ne peuvent être reliées qu'à des racines de S_2 par des arêtes de G.
4. Nous définissons deux ensembles X et Y composés de tous les sommets qui appartiennent à $S_1 \cup S_2$. Le premier ensemble X est composé de toutes les feuilles de S_2 et de tous les sommets de S_1, sa cardinalité est $\boldsymbol{|X| = (2N + K)}$. De plus, l'ensemble des sommets de X forme un stable aux vues des propriétés précédentes. Le second ensemble Y est composé de toutes les racines de S_2, et sa cardinalité est $\boldsymbol{|Y| = N}$. Y représente le voisinage de X, nous le notons $Y = \Gamma(X)$.
5. Pour tout 2-recouvrement M, le fait que $x \in S(M) \cap X \Rightarrow \exists e = \{x, y\} \in M$ et $y \in Y = \Gamma(X)$, entraîne que nous avons $\boldsymbol{|S(M) \cap X| \leq 2|Y|}$.

Maintenant, nous allons montrer que M_2 ne peut pas couvrir plus de sommets que M_1, et donc que $|M_2| = |M_1|$:

– Du fait que $|S(M_2)| > |S(M_1)|$ et que M_2 couvre K sommets non-saturés par M_1 dans S_1, M_2 ne couvre pas plus de $(K-1)$ sommets dans S_2. Donc M_2 doit couvrir au moins $|X|-(K-1) = 2N+1$ sommets de X. D'après les remarques précédentes, le nombre de sommets de X saturés par n'importe quelle couverture est inférieur à $2|Y|$. Ce qui est une contradiction puisque $|2N+1| > 2|Y|$. Nous en déduisons qu'il ne peut exister un 2-recouvrement M_2 tel que $|M_2| > |M_1|$. Donc nous avons $\boldsymbol{|M_2| = |M_1|}$, entraînant que M_1 est bien un 2-recouvrement maximum.

□

2.3.2 Algorithme en temps polynomial pour un 2-recouvrement maximum

En se basant sur les résultats du Théorème 2.3.1, nous pouvons présenter un algorithme donnant un 2-recouvrement maximum pour un graphe G quelconque. Soit M un 2-recouvrement, et soit C une chaîne M-alternée augmentante. L'algorithme échange les arêtes couvrantes à celles non-couvrantes dans C, à l'exception d'une des dernières arêtes de la chaîne selon les différents cas. Nous notons cette opération $\boldsymbol{Augmentation(M, C)}$, qui résulte en un nouveau 2-recouvrement couvrant un ou deux sommets de plus que M. L'algorithme généré par la définition d'un 2-recouvrement est donné dans l'**Algorithme 1**.

La recherche de l'existence d'une chaîne M-alternée augmentante C dans le graphe G va s'appliquer pour chaque sommet non-saturé x_0. Une première approche gloutonne serait de regarder toutes les chaînes M-alternées possibles en partant de x_0, puis de prendre la première qui est augmentante. Un algorithme plus efficace et bien plus rapide est basé sur l'algorithme du «parcours en largueur dans un arbre» où la racine est x_0. À partir de x_0, nous visitons les sommets voisins en largueur, selon l'appartenance alternée à M, et en cherchant un sommet qui sera à une distance impaire de x_0 et dont le degré relativement à M sera inférieur à deux. Si aucun sommet ne correspond, nous pouvons conclure à l'inexistence d'une chaîne M-alternée augmentante. La description est donné dans l'**Algorithme 2**.

Algorithme 1 : Recherche d'un 2-recouvrement maximum

Données : $G = (V, E)$
Résultat : Un 2-recouvrement M
début
  $M :=\emptyset$
  tant que *il existe une chaîne M-alternée augmentante C* **faire**
    $M := Augmentation(M, C)$
  Retourner M
fin

Algorithme 2 : Recherche d'une chaîne M-alternée augmentante

Données : $G = (V, E)$, avec $|V| = n$, un sommet x_0, et M un 2-recouvrement
Résultat : Une chaîne M-alternée augmentante C d'extrémité x_0
début
  Soit Q (resp. Z) une file d'attente dont l'unique élément est le sommet x_0 (resp. une file d'attente vide)
  Soit F une fonction qui prend un sommet en paramètre et donne le sommet qui le précède
  tant que $Q \neq \emptyset$ **faire**
    Soit u le premier élément de Q
    si $u \in Z$ **alors**
      Mettre dans Q les deux voisins de u selon l'appartenance à M
    sinon
      pour *chaque sommet v qui est voisin de u et tel que $v \notin Z$* **faire**
        $F[v] = u$
        si *v est un sommet se trouvant à une distance impaire de x_0 et tel que $d_M(v) < 2$* **alors**
          Retourner la chaîne M-alternée augmentante
          $C = \{x_0, \ldots, F(F(v)), F(v), v\}$
        sinon
          si *v est un sommet se trouvant à une distance impaire de x_0* **alors**
            Mettre v dans Z
          Mettre v dans Q
    Enlever u de Q
fin

L'**Algorithme 2**, basé sur un parcours en largeur, utilise une complexité de $\boldsymbol{O(n+m)}$ où n (resp. m) est le nombre de sommets (resp. d'arêtes). Dans l'**Algorithme 1**, dans le pire des cas, nous allons chercher n fois la présence d'une chaîne augmentante M-alternée. Ainsi l'**Algorithme 1** s'exécute au pire en $\boldsymbol{O(n(n+m))}$.

2.3.3 Problème dual du 2-recouvrement maximum

Nous finissons cette partie avec le problème dual du 2-recouvrement maximum :

Théorème 2.3.2 :
Soit S un ensemble indépendant de sommets dans un graphe G. Soit T (resp. K) l'ensemble (resp. le nombre) des sommets non saturés par un 2-recouvrement. Nous désignons par $N(S)$ l'ensemble des sommets voisins de l'ensemble S. Nous avons alors l'égalité suivante :

$$\max_{S} (|S| - 2|N(S)|) = \min_{T} K$$

Preuve :

- Montrons que $\max_{S}(|S| - 2|N(S)|) \leq \min_{T} K$:

  Soit S_0 l'ensemble qui maximise $|S_0| - 2|N(S_0)| = \alpha_0$.
  - Si $\alpha_0 = 0$, il est évident que $\max_{S}(|S| - 2|N(S)|) \leq \min_{T} K$.
  - Supposons maintenant que $\alpha_0 \geq 1$. Considérons le graphe biparti $B = (S_0, N(S_0), E_0)$ entre les sommets de S_0 et $N(S_0)$. Soit M un 2-recouvrement quelconque. Prenons le graphe B' induit par les arêtes de $M \cap E_0$.
    Dans le graphe B', $\forall x \in S_0$ nous avons $d_{B'}(x) = 0$ si x est non saturé, $d_{B'}(x) \geq 1$ si x est saturé, et $\forall x \in N(S_0)$ nous avons $d_{B'}(x) \leq 2$.
    Nous avons $S_0 = S_0^0 \cup S_0^1$ avec S_0^0 l'ensemble des sommets de S_0 isolés dans B' ($\forall x \in S_0^0, d_{B'}(x) = 0$), et S_0^1 l'ensemble complémentaire.
    Sachant que $|S_0^1| = \sum_{x \in S_0} d_{B'}(x) \leq \sum_{x \in N(S_0)} d_{B'}(x) \leq 2|N(S_0)|$. Ainsi, nous trouvons :

    $$\begin{aligned} |S_0| - |S_0^0| &\leq 2|N(S_0)| \\ |S_0| - 2|N(S_0)| &\leq |S_0^0| \\ \alpha_0 &\leq |S_0^0| \end{aligned}$$

    La relation est vraie pour n'importe quel 2-recouvrement et donc en particulier pour un 2−recouvrement maximum M_0. C'est à dire que le nombre de sommets isolés est minimisé par M_0, et donc $|S_0^0| = \min_{T} K$.

- Montrons que $\max_{S}(|S| - 2|N(S)|) \geq \min_{T} K$:

  Soit M_0 un 2−recouvrement maximum qui minimise le nombre de sommets isolés, et soit T_0 (resp. K) l'ensemble (resp. le nombre) des sommets non saturés.
  Nous considérons l'Algorithme 3 suivant :

Algorithme 3 : Algorithme donnant le stable S maximisant $|S| - 2|N(S)|$

$S_0 := T_0$; $V_0 := \varnothing$; $nonvisite := N(S_0)$
while $nonvisite \neq \varnothing$ **do**
  Choisir x dans l'ensemble $nonvisite$
  Soit $N_{M_0}(x)$ les voisins de x dans M_0
  $nonvisite := nonvisite - \{x\}$; $V_0 := V_0 \cup \{x\}$
  $\forall y \in N_{M_0}(x) : nonvisite := nonvisite \cup \{y\} - V_0$
  $S_0 := S_0 \cup N_{M_0}(x)$
end while

Cherchons les invariants à chaque itération de l'algorithme : l'ensemble des sommets de S_0 forme un stable, d'après le Lemme 2.3.1 nous avons $|N_{M_0}(x)| = 2$, ce qui entraîne que

$|S_0|$ augmente de 2. À chaque itération nous augmentons également l'ensemble V_0 de 1, et les invariants précédents entraîne l'égalité suivante $|S_0| - 2|V_0| = K$ à chaque étape.
À la fin de l'algorithme on a donc $|S_0| - 2|V_0| = K$ où S_0 est un stable et $V_0 = N_G(S_0)$, ce qui entraîne que $|S_0| - 2|N_G(S_0)| = K$ avec S_0 qui forme un stable. Donc nous avons forcément $\max_S |S| - 2|N(S)| \geq K$.

□

2.4 Conclusion

Dans ce chapitre nous avons présenté les différentes notions nécessaires pour la suite du manuscrit, regroupant les éléments de la théorie des ensembles, des graphes, de l'ordonnancement, de la complexité et de l'approximation. Puis nous avons présenté différents problèmes connus de la littérature utilisés dans la suite. Enfin ce chapitre finit par l'introduction d'un type de recouvrement de sommets dans un graphe, le 2-recouvrement. Nous avons donné plusieurs définitions et propriétés propres à cette technique, puis les algorithmes pour l'obtenir en temps polynomial et enfin le dual du problème. Nous allons maintenant présenter le problème général de la torpille, le modéliser et donner un état de l'art portant sur des problèmes particuliers.

Chapitre 3

Modélisation et état de l'art

3.1 Introduction

L'utilisation d'une torpille totalement autonome est assez récente, son étude requiert une étude théorique solide sur laquelle les roboticiens peuvent se baser pour mieux appréhender l'utilisation qu'ils feront de cette torpille. Les roboticiens décident à l'avance des tâches que devra exécuter la torpille lorsqu'elle sera en mer, lui donnant ainsi un ordre de précédence selon les actions et les temps des différentes tâches à réaliser. Toutes les données sont donc connues et utilisables par l'ordonnanceur embarqué, seuls les capteurs acoustiques risquent de poser problème du fait qu'ils utilisent des ondes se propageant sous l'eau et rebondissant sur des parois. Le temps de propagation de ces ondes n'est pas connu à l'avance vu que la torpille se déplace à l'aveugle sous l'eau, mais ils ont décidé pour pouvoir simplifier leur calcul de définir des bornes supérieures. Ainsi passé un délai fixé, pour chaque capteur envoyant une onde, l'onde n'est plus prise en compte et les données enregistrées sont des données indiquant une trop grande distance d'une paroi. Dans la suite nous considérons le problème où il y a toujours un retour de l'onde (il sera intéressant dans une prochaine étude d'étudier le cas lorsqu'il n'y a pas de retour et donc une indication de mauvaise capture. Le but sera donc de minimiser le nombre de re-émission). Donc les temps de propagation seront toujours donnés à l'avance grâce à ces bornes supérieures.

Les premières expérimentations ont permis de vérifier si les différents capteurs mis en place sur TAIPAN fonctionnaient parfaitement, et révélant ainsi l'ensemble des capteurs ne pouvant être utilisés en même temps pour cause d'interférences dues à leur proximité et leur fréquence trop proche l'une de l'autre. Les roboticiens définissent alors une liste de couples de capteurs pouvant être utilisés en même temps.

Nous avons décidé d'aborder le problème de façon rigoureuse et la plus complète possible pour l'ordonnancement sur mono processeur de tâches d'acquisition avec présence ou non de tâches de traitement et de contraintes d'incompatibilité. La modélisation que nous allons donner est celle utilisée tout au long du manuscrit, où nous avons une connaissance totale des données et des paramètres avant de commencer à ordonnancer les tâches. Cette étude va permettre de pouvoir nous positionner par rapport aux travaux déjà existants dans la littérature pour ce genre de tâches nommées **tâches couplées** (**coupled-tasks** en anglais).

3.2 Concept et modèle

La torpille TAIPAN reçoit plusieurs instructions, appelées **sous-objectifs** par les roboticiens, comme par exemple vérifier l'état d'un pipeline, cartographier les fonds sous-marins, mesurer la salinité de l'eau, etc... Chacun de ces sous-objectifs est pris en compte par différents

modules gérant l'instrumentation (capteurs et actionneurs) et des modules effectuant des traitements informatiques sur les données récupérées par ces derniers. L'exécution d'un module sur le processeur se traduira par une tâche à exécuter par l'**ordonnanceur** sur le processeur (voir illustration Figure 3.1).

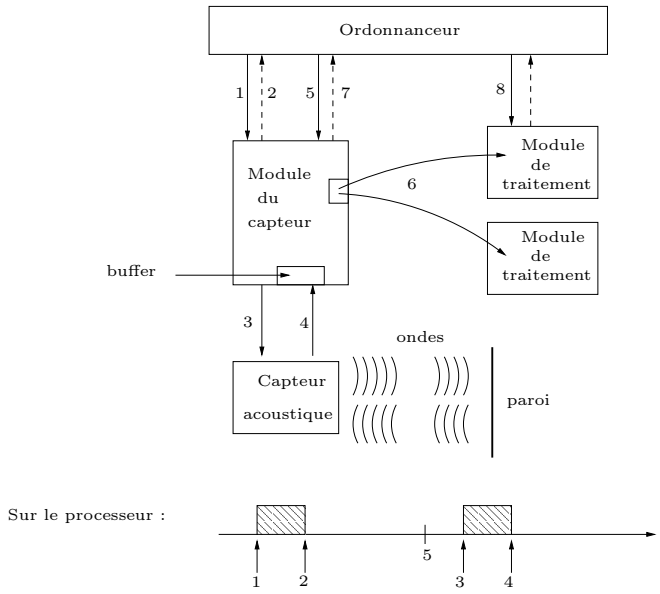


Fig. 3.1 – Illustration de l'activation d'un module gérant la captation acoustique pour récupérer des données. L'ordonnanceur décide de procéder à une acquisition, en (1) un signal est envoyé au module du capteur pour lui demander de lancer l'onde, celui-ci va réaliser plusieurs calculs nécessitant l'utilisation du processeur pour tout mettre en place et une fois prêt, il émet un message en (2) pour signaler à l'ordonnanceur que le capteur va envoyer l'onde. Une fois l'envoi de cette information, un signal est émis au capteur en (3) pour lui ordonner d'envoyer l'onde (remarquons que cette étape ne nécessite pas l'utilisation du processeur). Durant la propagation de l'onde, le processeur reste inactif. Une fois l'onde revenue au capteur, celui-ci transmet en (4) les données au module où elles seront stockées dans un tampon. Après un certain temps passé, l'ordonnanceur communique au module en (5) pour lui demander de récupérer les données et les traiter. Le module va réaliser différents calculs nécessaires comme par exemple le recodage des données, et va transférer en (6) les données stockées dans le tampon vers les modules nécessitant ces données pour différents calculs. Cette phase nécessite l'utilisation du processeur et s'arrête lorsque, une fois cette action finie, un signal de fin est envoyé en (7) à l'ordonnanceur. Une fois cette tâche finie, l'ordonnanceur prend la suivante selon l'ordre qu'il trouve, et recommence en (8) le même processus qu'il s'agisse d'un module de traitement ou d'acquisition (les modules de traitement réaliseront seulement des calculs et des traitements selon les données qu'ils ont reçues).

Si nous prenons l'exemple du sous-objectif qui consiste à vérifier l'état d'un pipeline, nous allons avoir quatre modules : repérage GPS, navigation vers le pipeline, et inspection du pipeline. Ces modules doivent respecter leur ordre d'exécution. Il se peut que plusieurs sous-objectifs soient envoyés à la torpille, dans ce cas les modules peuvent avoir lieu en même temps. Si nous reprenons l'exemple précédent et que nous ajoutons le sous-objectif de cartographier le fond sous-marin, ce sous-objectif peut être réalisé en même temps que d'autres sous-objectifs. Mais le risque d'interférence (dû la plupart du temps par la forte proximité de certains capteurs acoustiques ou d'une même fréquence utilisée), qui se répercuteront en erreurs de mesures sur les données acquises, entraîne la nécessité de prévoir les modules qui peuvent avoir lieux en même temps.

Nous allons maintenant donner la modélisation du problème de la torpille du point de vue de l'ordonnancement, avec une formalisation des paramètres, des notations des différentes tâches et contraintes que nous utiliserons dans la suite du manuscrit.

3.2.1 Description des tâches d'acquisition et des tâches de traitement

L'ensemble des sous-objectifs défini par les roboticiens est représenté par un groupe de modules comme nous l'avons vu précédemment. Pour un sous-objectif donné l'ensemble de ses modules peut être assimilé à différentes tâches qui devront être exécutées sur le processeur. Nous avons une partition des tâches à exécuter en deux ensembles :

- **Les tâches d'acquisition**
- **Les tâches de traitement**

L'activation et l'utilisation d'un module de **capteur acoustique** gérant un moyen de perception (voir Figure 3.1) sera appelé tâche d'acquisition et aura la forme d'une tâche-couplée.

Définition 3.2.1 (Tâche d'acquisition) :
Soit $\boldsymbol{\mathcal{A} = \{A_1, A_2, \ldots, A_k\}}$ *l'ensemble des* ***tâches d'acquisition*** *à exécuter,* $k \in \mathbb{N}^*$. *Soit* $\boldsymbol{A_i \in \mathcal{A}}$ *une* ***tâche d'acquisition****,* $1 \leq i \leq k$, *représentée par une* ***tâche couplée****.* A_i *est formée de deux sous-tâches* $\boldsymbol{a_i}$ *et* $\boldsymbol{b_i}$ *séparées par un temps d'inactivité incompressible et indilatable de longueur* $\boldsymbol{L_i}$. *Pour une simplification des notations, les temps d'exécution des sous-tâches seront également notés* a_i *et* b_i *et non* p_{a_i} *et* p_{b_i} *de manière classique. Ainsi le temps total d'une tâche d'acquisition sera* $(a_i + L_i + b_i)$, *le temps de début d'exécution* $\sigma(A_i) = \sigma(a_i)$ *et la date de complétion* $C(A_i) = \sigma(a_i) + a_i + L_i + b_i$ *(voir Figure 3.2).*

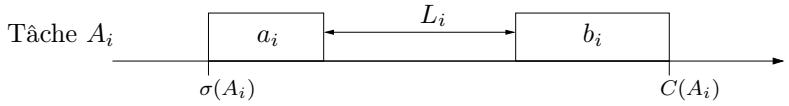


Fig. 3.2 – Illustration d'une tâche d'acquisition A_i

Par la suite, nous décrirons les tâches d'acquisition par le **triplet** $\boldsymbol{(a_i, L_i, b_i)}$. La sous tâche a_i (resp. b_i) représente le temps d'utilisation du processeur entre le moment 1 et 2 (resp. 3 et 4) sur la Figure 3.1. La variable L_i représente le temps d'inactivité entre les moments 2 et 3 sur la Figure 3.1.

Définition 3.2.2 (Slot d'inactivité) :
Un ***slot*** *est un intervalle de temps d'inactivité sur le processeur. Nous utiliserons cette appellation pour nommer le temps d'inactivité créé par une tâche d'acquisition.*

Pendant le slot d'inactivité d'une tâche d'acquisition, une autre tâche peut être exécutée (le processeur étant libre) mais doit s'arrêter avant l'exécution de la deuxième sous-tâche. En effet la deuxième sous-tâche ne peut être préemptée ni retardée puisqu'elle doit se synchroniser avec l'instrumentation.

Lorsque la torpille devra juste exécuter un module contenant un **simple traitement informatique**, nous parlerons alors de tâche de traitement.

Définition 3.2.3 (Tâche de traitement) :
Soit $\boldsymbol{\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_l\}}$ *l'ensemble des* ***tâches de traitement*** *à exécuter,* $l \in \mathbb{N}^*$. *Soit* $\boldsymbol{\tau_i \in \mathcal{T}}$ *une* ***tâche de traitement****,* $1 \leq i \leq l$, $\boldsymbol{\tau_i}$ *est une tâche préemptive avec un temps d'exécution* $\boldsymbol{\tau_i}$ *pour les simplifications de notation. Les tâches de traitement seront toujours représentées sur les illustrations par un trapèze de manière à les différencier des tâches d'acquisition.*

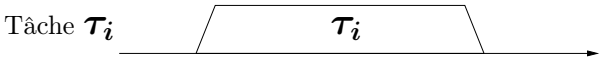


Fig. 3.3 – Illustration d'une tâche de traitement $\mathcal{T}_i$

L'exécution d'un sous-objectif va consister en l'exécution d'une suite de tâches dans un ordre précis, ce qui implique d'avoir une relation d'ordre d'un ensemble de tâches modélisé par un graphe orienté nommé graphe de précédence. Ces graphes représentent l'ensemble des sous-objectifs à réaliser.

Définition 3.2.4 (Graphe de précédence) :
Soit $\boldsymbol{\mathcal{GP} = \{GP_1, GP_2, \ldots, GP_p\}}$ *l'ensemble des* ***graphes de précédence*** *à exécuter,* $p \in \mathbb{N}^*$*. Soit* $\boldsymbol{GP_i = (V_i, E_i)} \in \boldsymbol{\mathcal{GP}}$*,* $1 \leq i \leq p$*, un* ***graphe de précédence*** *où* $\boldsymbol{V_i}$ *est un ensemble de tâches d'acquisition et de traitement, et* $\boldsymbol{E_i}$ *représente l'ensemble des arcs reliant deux tâches qui ont une relation de précédence. L'ensemble des tâches d'acquisition de* V_i *seront toujours les sources des graphes* GP_i*, et de plus nous pourrons avoir plusieurs tâches d'acquisition comme prédécesseurs d'une seule tâche de traitement mais jamais une tâche d'acquisition sans successeur.*

Les tâches d'acquisition vont avoir un autre type de contraintes comme nous l'avons vu précédemment, certaines tâches ne pourront avoir lieu en même temps afin d'éviter les interférences lors de la captation des données. Afin de modéliser au mieux les couples qui sont compatibles ou non, nous définissons un graphe de compatibilité.

Définition 3.2.5 (Graphe de compatibilité) :
Soit $\boldsymbol{G_c = (\mathcal{A}, E_c)}$ *un* ***graphe de compatibilité*** *où les sommets représentent l'ensemble des tâches d'acquisition* $\boldsymbol{\mathcal{A}}$*, et* $\boldsymbol{E_c}$ *représente l'ensemble des arêtes reliant deux tâches d'acquisition qui sont* ***compatibles*** *(voir illustration Figure 3.4).*

Définition 3.2.6 (Tâches d'acquisition compatibles) :
Deux tâches d'acquisition $\boldsymbol{A_i}$ *et* $\boldsymbol{A_j}$ *sont dites* ***compatibles*** *lorsque leurs temps d'inactivité respectifs* L_i *et* L_j *peuvent avoir lieu en même temps (voir illustration Figure 3.4). Les tâches* A_i *et* A_j *peuvent alors se chevaucher lors de leur exécution.*

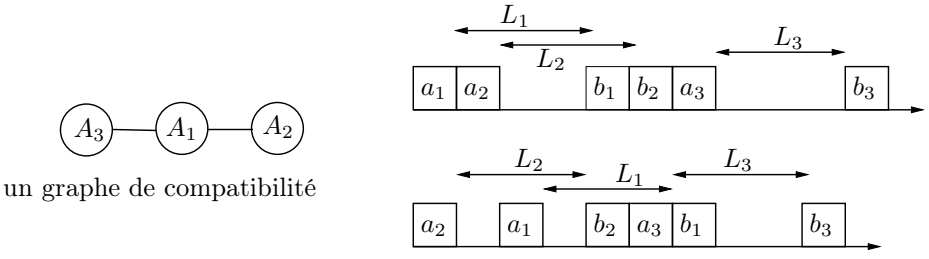


Fig. 3.4 – Illustration de la contrainte d'incompatibilité

3.2.2 Présentation et description du modèle

Nous allons utiliser le formalisme habituel de la théorie de l'ordonnancement pour simplifier les notations, et décrire les différentes contraintes sous forme d'équations mathématiques. De manière formelle, notre problème peut être défini ainsi :

- $\boldsymbol{\sigma(b_i) = \sigma(a_i) + a_i + L_i}$, $\forall A_i \in \mathcal{A}$ où $\sigma(a_i)$ désigne la date de début d'exécution de la sous-tâche a_i[4] (voir Définition 2.1.12)
- $\boldsymbol{\sigma(\tau_i) \geq C(b_j)}$ (resp. $\boldsymbol{\sigma(\tau_j) \geq C(\tau_i)}$), $\forall A_j$,τ_i (resp. $\forall \tau_i$,τ_j) tel qu'il existe une contrainte de précédence entre A_j and τ_i (resp. entre τ_i et τ_j). $C(b_j)$ représente la date de fin d'exécution de la sous-tâche b_j (voir Définition 2.1.13).
- $\boldsymbol{\sigma(a_i) \notin [\sigma(a_j), C(b_j)[}$ et $\boldsymbol{\sigma(b_i) \notin [\sigma(a_j), C(b_j)[}$, $\forall i$, $\forall j$ tels que A_i et A_j sont incompatibles.

En reprenant la notation $\boldsymbol{\alpha|\beta|\gamma}$ suggérée par Graham [37] et Blazewicz [12] (voir la Section 2.1.3 page 9), notre problème peut être décrit de la manière suivante :

$$\boldsymbol{1|prec, tâche-couplée, (a_i, L_i, b_i) \cup (\tau_i, pmtn), G_c|C_{max}} \tag{3.1}$$

c.-à-d. le paramètre $\boldsymbol{\alpha}$ sera toujours égal à 1 vu que nous travaillerons exclusivement sur un mono-processeur, d'ailleurs dans la suite nous dirons simplement "processeur" au lieu de "mono-processeur" pour alléger les appellations. Le paramètre $\boldsymbol{\beta}$ va souvent varier, précisant lorsque nous aurons des contraintes de précédence quelconque avec *prec*, le type de tâches d'acquisition seront décrites par le triplet (a_i, L_i, b_i), le type de tâches de traitement par τ_i pour la durée des tâches et *pmtn* pour l'indication de préemptivité, et enfin la contrainte d'incompatibilité sera indiquée par la présence ou non de G_c. Le paramètre $\boldsymbol{\gamma}$ décrira les différents objectifs que nous étudierons, ce sera la plupart du temps la minimisation du temps de complétion C_{max} (voir Définition 2.1.14).

3.3 État de l'art

Depuis plus de cinquante ans, la théorie de l'ordonnancement a été largement étudiée, depuis les premiers schémas d'ordonnancement en 1967 par Conway, Maxwell et Miller [22], en passant par Lenstra, Lawler et Rinnooy Kan en 1982 [48] qui redéfinissent ces schémas. Ils ont défini la plupart des objectifs que l'on peut rencontrer en ordonnancement, les différentes contraintes possibles pour des tâches, et surtout une hiérarchie de la complexité des problèmes d'ordonnancement selon les objectifs étudiés. Dans la suite nous nous intéresserons uniquement à la minimisation du makespan $\boldsymbol{C_{max}}$ et à la minimisation de la somme des dates de complétion $\boldsymbol{\sum C(i)}$.

Nous allons présenter dans une première section, les principaux résultats classiques des problèmes d'ordonnancement sur mono processeur, présentant ainsi les différentes contraintes et objectifs connus, et enfin les méthodes utilisées pour obtenir une solution optimale ou approchée de ces problèmes.

Nous focaliserons ensuite notre étude sur les tâches-couplées qui sont au cœur de notre problématique. La structure particulière de ces tâches amène des problèmes d'ordonnancement très spécifiques, nous allons donner un état de l'art sur l'étude de la complexité et de l'approximation de ces problèmes sur mono processeur. Après une présentation des domaines d'application, nous donnerons une vision globale et hiérarchique des résultats de complexité se trouvant dans la littérature, puis nous parlerons des techniques utilisées pour résoudre ou approcher ces problèmes selon leur complexité.

A partir de là, nous nous positionnerons par rapport à ces travaux, en donnant les grandes lignes de notre motivation pour la suite qui consiste à mesurer l'impact des contraintes d'incompatibilité et de précédences. Ces graphes entraîneront une analyse différente pour l'étude de la complexité et de l'approximation de nos problèmes, une étude qui portera essentiellement

[4] Nous rappelons que la durée d'exécution de la sous-tâche a_i est notée a_i et non p_{a_i}.

sur le recouvrement de sommets dans un graphe. Nous présenterons un état de l'art sur les techniques et problèmes connus de recouvrement de sommets dans un graphe. Nous focaliserons notre état de l'art à quatre problèmes en particulier, en donnant pour chacun d'eux une vision de la complexité selon le type de graphe que l'on peut avoir.

3.3.1 État de l'art sur les problèmes d'ordonnancement sur mono processeur

Les mono processeurs sont des environnements simples permettant de mettre en avant la difficulté de certains paramètres choisis ou d'objectifs étudiés, et ainsi connaître les difficultés intrinsèques des problèmes. Les problèmes sur multi-processeurs engendrent une multitude de nouveaux paramètres à prendre en compte comme les communications, les distances entre les processeurs, les temps de calcul différents selon les processeurs, etc. Les heuristiques qui sont efficaces pour des problèmes sur mono processeur donnent une première idée de la manière d'aborder les mêmes problèmes sur des environnements plus complexes comme les multi-processeurs.

Étant à l'origine des problèmes d'usines, des problèmes très concrets, la **théorie de l'ordonnancement** sur mono processeur s'est très rapidement développée selon les différents types de problèmes que l'on pouvait rencontrer. En partant d'un simple problème d'ordonnancement, considérant un ensemble de tâches quelconques non préemptives et dont l'objectif est de trouver la longueur d'ordonnancement la plus petite possible, une multitude de problèmes ont été étudiés en ajoutant des paramètres. Certains de ces paramètres se retrouvent dans notre problématique et nous nous sommes intéressés aux différentes approches utilisées.

Les problèmes $1 \mid \beta \mid C_{max}$

Les travaux réalisés dans la littérature sont souvent regroupés selon les objectifs étudiés. Ainsi, pour tout problème ayant pour objectif l'optimisation du makespan C_{max}, nous trouvons dans la littérature en ordonnancement [50, 14, 32, 6, 15, 28, 59, 60, 46, 18] l'étude et la classification de la plupart des problèmes classiques d'ordonnancement. Pour chaque problème étudié, les auteurs font varier les différents paramètres du champs β.

Parmi les résultats classiques de la théorie de l'ordonnancement sur mono processeur, nous donnerons ceux utilisant des caractéristiques simples. Lenstra et al. montrent dans [50] la $\mathcal{NP}$-complétude du problème $\boldsymbol{1|r_j, \tilde{d}_j|C_{max}}$, le paramètre $\boldsymbol{r_j}$ représentant les dates minimales de début d'exécution des tâches et $\boldsymbol{\tilde{d}_j}$ les dates maximales de fin d'exécution des tâches. Pour le cas où toutes les tâches ont un temps d'exécution unitaire $\boldsymbol{1|r_j, p_j = 1, \tilde{d}_j|C_{max}}$, Garey et al. [32] résolvent ce problème en donnant un algorithme en temps polynomial utilisant la technique d'évaluation et séparation. Les mêmes auteurs montrent qu'en rajoutant des contraintes de précédence entre les tâches (noté $\boldsymbol{prec}$ dans β), la complexité des deux problèmes cités précédemment ne change pas. Enfin, parmi les problèmes classiques ayant pour objectif la minimisation du C_{max}, Lawler [47] donne un algorithme en temps polynomial pour résoudre le problème $\boldsymbol{1|prec, r_j|C_{max}}$.

Il existe bien d'autres problèmes avec pour objectif $\boldsymbol{C_{max}}$, utilisant des caractéristiques très différentes, mais dans un premier temps, nous avons choisi de rappeler les résultats utilisant les contraintes classiques $\boldsymbol{prec, r_j}$ et $\boldsymbol{\tilde{d}_j}$ (voir Tableau 3.1).

Les problèmes $1 \mid \beta \mid \{\sum_j C(j), \sum_j w_j C(j)\}$

La somme des temps de complétude est souvent utilisée pour mesurer le temps de réponse des différentes tâches ou encore mesurer le temps de présence des tâches dans le système.

Problème	Complexité	Référence
$1\|prec\|C_{max}$	Polynomial	Finta (1996) [27]
$1\|prec, r_j\|C_{max}$	Polynomial	Lawler (1973) [47]
$1\|r_j, \tilde{d}_j\|C_{max}$	$\mathcal{NP}$-complet	Lenstra (1977) [50]
$1\|prec, r_j, \tilde{d}_j\|C_{max}$	$\mathcal{NP}$-complet	Garey (1981) [32]
$1\|r_j, p_j = 1, \tilde{d}_j\|C_{max}$	Polynomial	Garey (1981) [32]
$1\|prec, r_j, p_j = 1, \tilde{d}_j\|C_{max}$	Polynomial	Garey (1981) [32]

TAB. 3.1 – Récapitulatif de quelques résultats d'ordonnancement avec C_{max}

Lorsque nous voulons mettre l'accent sur le coût que peut représenter l'attente de fin d'exécution des tâches, un paramètre de poids de pénalité w_j est ajouté à chaque tâche et l'objectif est $\sum w_j C(j)$. Nous allons présenter les différents résultats selon que l'objectif soit $\sum \boldsymbol{C(j)}$ ou $\sum \boldsymbol{w_j C(j)}$.

Les deux problèmes de base $1||\sum w_j C(j)$ et $1||\sum C(j)$ peuvent être résolus de manière optimale selon des règles d'ordonnancement :

- Le premier problème peut être résolu de manière optimale par une des règles les plus connues dans la théorie de l'ordonnancement, la règle de Smith [64] Weighted Shortest Processing Time first (**WSPT** rule) qui consiste à ordonnancer les tâches de manière décroissante selon le ratio p_j/w_j.
- Pour le second problème, qui n'est d'autre que le cas particulier où $w_j = 1$, l'algorithme donnant une solution optimale est basé sur la règle Shortest Processing Time (**SPT** rule).

Lenstra et al. démontrent dans [50] la $\mathcal{NP}$-complétude des mêmes problèmes en présence de dates de début d'exécution r_j. En ajoutant le paramètre de préemptivité $pmtn$, Baker [5] propose une extension de la règle SPT de Smith [64] pour résoudre, via un algorithme polynomial, de manière optimale $1|pmtn, r_j|\sum C(j)$, alors que Labetoulle et al. montrent dans [45] la $\mathcal{NP}$-complétude de $1|pmtn, r_j|\sum_j w_j C(j)$.

Notons qu'en ajoutant des dates de fin maximale d'exécution, les résultats sont à peu près similaires selon les paramètres ajoutés. Et enfin pour ce qui est des contraintes de précédence, Lenstra et al. prouvent dans [49] la $\mathcal{NP}$-complétude de $1|prec|\sum w_j C(j)$ et même lorsque $w_j = 1$ et $p_j = 1$. Mais avec la présence de dates minimales de début d'exécution r_j, la plupart de ces problèmes deviennent polynomiaux.

Les deux objectifs $\sum w_j C(j)$ et $\sum C(j)$ ont été largement étudiés depuis ces premiers résultats, utilisant des caractéristiques très différentes. Mais dans un premier temps, nous avons choisi de rappeler les résultats utilisant les contraintes classiques $prec, r_j$ et $pmtn$ (voir Tableau 3.2).

Ce rapide survol des résultats de complexité en ordonnancement sur mono processeur nous permet de mieux situer la difficulté des contraintes que nous pouvons avoir pour notre problème d'ordonnancement lié à l'acquisition de données par une torpille en immersion. Ces contraintes vont bien sûr jouer un rôle important sur l'étude de notre problème mais le type très particulier des tâches à exécuter va nous pousser à réfléchir différemment. Nous avons vu que les tâches d'acquisition étaient en fait connues dans la littérature sous le nom de tâches-couplées (coupled-task en anglais), dans la section suivante nous présentons les différents problèmes et résultats existants sur les problèmes d'ordonnancement en présence de tâches-couplées.

Problème	Complexité	Référence
$1\|r_j\|\sum C(j)$	$\mathcal{NP}$-complet	Lenstra et al. (1977) [50]
$1\|r_j\|\sum w_jC(j)$	$\mathcal{NP}$-complet	Lenstra et al. (1977) [50]
$1\|pmtn, r_j\|\sum C(j)$	Polynomial	Baker (1974) [5]
$1\|pmtn, r_j\|\sum w_jC(j)$	$\mathcal{NP}$-complet	Labetoulle et al. (1984) [45]
$1\|prec\|\sum C(j)$	$\mathcal{NP}$-complet	Lawler (1978) [49]
$1\|prec\|\sum w_jC(j)$	$\mathcal{NP}$-complet	Lawler (1978) [49]
$1\|prec, p_j = p, r_j\|\sum C(j)$	Polynomial	Simons (1983) [62]

TAB. 3.2 – Récapitulatif de quelques résultats d'ordonnancement avec pour critère $\sum C(j)$ et $\sum w_jC(j)$

3.3.2 État de l'art sur les tâches-couplées

Les problèmes d'ordonnancement en présence de tâches-couplées sont étudiés dans la littérature depuis presque 30 ans, ce terme a été introduit la première fois par Shapiro [61] afin de modéliser des problèmes de transmissions par ondes d'un radar. L'objectif est la plupart du temps de minimiser le temps de fin d'exécution de la dernière tâche sur le processeur, l'objectif théorique est alors souvent modélisé par le makespan C_{max}.

La plupart des travaux portant sur l'étude et l'utilisation des tâches-couplées ont souvent été motivés par des problèmes d'émission de signal radar dans un environnement militaire comme le guidage d'arme ou le traçage d'une cible [55]. Mais le progrès technologique a permis à la robotique de développer des appareils utilisant des impulsions magnétiques comme par exemple pour la torpille TAIPAN (pour plus d'information sur le fonctionnement de la torpille TAIPAN, voir la thèse en robotique de El Jalaoui [40]). Ce développement a permis d'augmenter l'utilisation en ordonnancement des tâches-couplées, donnant ainsi de nouveaux types de problèmes avec des tâches-couplées, entraînant une étude plus théorique focalisée sur les différents types de tâches-couplées que l'on peut obtenir en faisant varier les **trois paramètres fondamentaux** d'une tâche-couplée : $\boldsymbol{(a_i, L_i, b_i)}$.

Complexité

L'étude la plus complète qui ait été faite sur les problèmes d'ordonnancement sur mono processeur avec tâches-couplées est dû à **Orman et Potts** [57], qui ont dénombré la plupart des problèmes avec tâches-couplées où l'objectif est de minimiser le makespan C_{max}, puis ont hiérarchisé ces problèmes selon leur complexité et ainsi selon leur difficulté algorithmique. Leur motivation vient d'un problème de radar à impulsions magnétiques permettant de localiser un objet en le traquant ou encore de surveiller un volume d'air en attendant qu'un événement se produise [56]. Orman et Potts [57] ont cherché à déterminer, selon les paramètres fondamentaux, la frontière entre le caractère polynomial et la $\mathcal{NP}$-complétude de ces problèmes d'ordonnancement avec des tâches-couplées.

Dans l'article les auteurs considèrent n tâches-couplées et étudient tous les cas possibles selon la durée des temps d'exécution des deux sous-tâches (a_i et b_i) et du temps d'inactivité L_i. Dans leur étude ils ne considèrent aucune contrainte d'incompatibilité, nous pouvons donc supposer que le graphe de compatibilité est complet. Pour simplifier la notation des problèmes, ils définissent les différents problèmes avec tâches-couplées par le triplet (a_i, L_i, b_i) où chaque paramètre peut varier. Le Tableau 3.3 synthétise leurs résultats en montrant la $\mathcal{NP}$-complétude de quatre problèmes en particulier et la polynomialité de deux autres :

À partir de ces résultats, ils caractérisent la complexité de différents problèmes plus généraux

Problème	Complexité
$a_i = L_i = b_i$	$\mathcal{NP}$-complet
$a_i = a, L_i, b_i = b$	$\mathcal{NP}$-complet
$a_i = a, L_i = L, b_i$	$\mathcal{NP}$-complet
$a_i = b_i = p, L_i$	$\mathcal{NP}$-complet
$a_i = b_i = p, L_i = L$	Polynomial
$a_i = L_i = p, b_i$	Polynomial

TAB. 3.3 – Résultats de complexité prouvés par **Orman et Potts** [57]

que ceux $\mathcal{NP}$-complets ou plus spécifiques que ceux polynomiaux. Une visualisation globale de la complexité de tous les résultats obtenus est présentée sur la Figure 3.5. Les treillis contenant les différents problèmes sont décrits de la manière suivante :

- (a_i, L_i, b_i) indique le type de problème d'ordonnancement avec tâches d'acquisition
- $(a_i = L_i, b_i)$ et les formes similaires indiquent les problèmes où b_i est non restreint et où $a_i = L_i,\ \forall i = 1, \ldots, n$
- $(a_i = a, L_i, b_i)$ et les formes similaires indiquent les problèmes où b_i et L_i sont non restreints et où $a_i = a,\ \forall i = 1, \ldots, n$
- $(a_i = L_i = p, b_i)$ et les formes similaires indiquent les problèmes où b_i est non restreint et où $a_i = L_i = p,\ \forall i = 1, \ldots, n$, pour un certaine constante $p \in \mathbb{N}^*$.
- Enfin les arcs sont orientés d'un cas spécifique vers un problème plus général, et les arêtes relient deux problèmes symétriques.

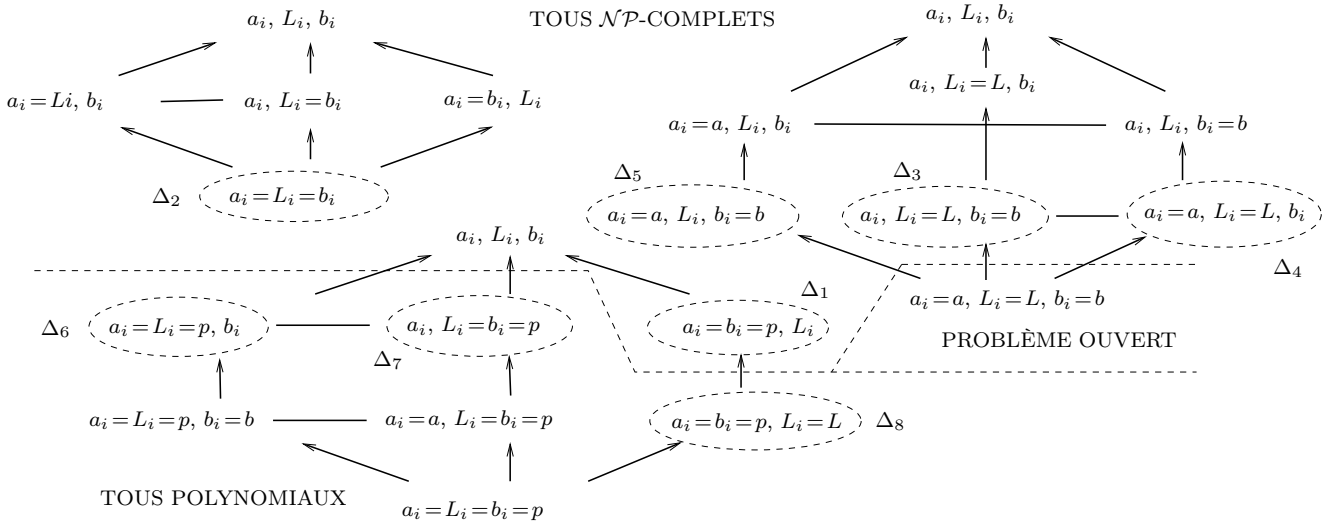


FIG. 3.5 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches-couplées et graphe de compatibilité complet où l'objectif est de minimiser le makespan (Orman et Potts [57]). Ils prouvent dans [57] la $\mathcal{NP}$-complétude des problèmes $\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5$ et la polynomialité des problèmes $\Delta_6, \Delta_7, \Delta_8$, la complexité des autres problèmes se déduit naturellement.

Les travaux d'Orman et Potts donnent une excellente vision de l'échelle de complexité de ces problèmes selon les paramètres fondamentaux. Ils laissent néanmoins un problème ouvert noté $\boldsymbol{(a_i = a, L_i = L, b_i = b)}$, il y a eu peu d'études sur ce cas dont le papier de Ahr et al. [2] qui proposent un algorithme exact utilisant la programmation dynamique de complexité

$O(nr^{2L})$ où $r \leq \sqrt[a-1]{a}$, tel que l'algorithme est linéaire en nombre de tâches pour un L fixé. Les auteurs définissent une technique de représentation utilisant des patterns composés de 1 et de 0 et représentant respectivement les temps d'utilisation du processeur et les temps d'inactivité. Ils décrivent alors, selon les valeurs de a et b, des graphes orientés pondérés représentant tous les patterns possibles. Puis, l'algorithme parcourt les différents patterns selon les arcs pour obtenir le poids minimum représentant le makespan. Cette programmation dynamique permet de résoudre le problème pour des petites instances, ou lorsque L est fixé. Brauner et al. [16] ont adapté cet algorithme pour résoudre un problème de tâches-couplées motivé par des problèmes de gestion de temps de production cyclique avec des robots.

Résultats d'approximation

Pour ce qui est de l'approximation de ces problèmes, plusieurs auteurs s'y sont intéressés selon des cas bien précis motivés par les problèmes modélisés. Ainsi Ageev et Baburin [26] proposent une $\frac{7}{4}$-approximation pour résoudre le problème $\mathcal{NP}$-complet $\boldsymbol{(a_i = b_i = 1, L_i)}$ (noté Δ_1 dans la Figure 3.5). Cet algorithme peut être implémenté en $\boldsymbol{O(n\ log\ n)}$ et consiste à construire deux ordonnancements différents tous les deux basés sur un algorithme classique d'ordre de priorité des tâches à exécuter. L'algorithme choisit alors l'ordonnancement le plus court des deux.

Pour le problème $\mathcal{NP}$-complet $\boldsymbol{(a_i, L_i, b_i)}$, Ageev et Kononov [1] donnent plusieurs résultats d'approximation et des bornes de non-approximabilité selon les valeurs de a_i et b_i. Les auteurs donnent un seul algorithme qu'ils utilisent pour chaque cas étudié, cet algorithme consiste à trier par ordre décroissant les tâches-couplées selon le paramètre L_i, puis à construire des blocs de tâches-couplées qui seront exécutés dans l'ordre. L'analyse des performances relatives est basée sur une analyse des temps d'inactivité obtenus pour chaque bloc. Cet algorithme peut être implémenté en $\boldsymbol{O(n\ log\ n)}$. Ils finissent le papier en donnant des bornes de non approximation pour chaque cas, les preuves sont similaires et sont basées sur une réduction à partir du problème $3 - PARTITION$. L'ensemble des résultats obtenus est présenté dans le Tableau 3.4 :

Problème	Ratio d'approx.	Borne de non approx.	Complexité
a_i, L_i, b_i	$\frac{7}{2}$	$2 - \epsilon$	$O(n\ log\ n)$
$a_i \leq b_i, L_i$	3	$2 - \epsilon$	$O(n\ log\ n)$
$a_i \geq b_i, L_i$	3	$2 - \epsilon$	$O(n\ log\ n)$
$a_i = b_i, L_i$	$\frac{5}{2}$	$2 - \epsilon$	$O(n\ log\ n)$

TAB. 3.4 – Résultats d'approximation et non-approximabilité donnés par Ageev et Kononov [1]

Peu de travaux ont été réalisés en rajoutant des contraintes aux tâches-couplées, mais nous pouvons noter deux résultats intéressants. Le premier est donné par Blażewicz et al. [11], et concerne le problème polynomial $\boldsymbol{(a_i = b_i = 1, L_i = L)}$ (noté Δ_8 dans la Figure 3.5). Ils démontrent le caractère $\mathcal{NP}$-complet du problème lorsque des contraintes de précédence sont ajoutés entre les tâches-couplées. La preuve est assez complexe et nécessite une double réduction polynomiale en partant du problème de décision $3 - PARTITION$ (Problème 2.2.9).

Le deuxième résultat porte sur une **étude on-line** des problèmes avec tâches-couplées où les auteurs Orman et al. [55] rajoutent des dates limites de fin d'exécution pour chaque tâche-couplée. Il y a eu peu d'études dans la littérature sur les problèmes d'ordonnancement on-line avec tâches-couplées. Ce papier présente plusieurs heuristiques consistant à trier les tâches de telle sorte qu'elles finissent leur exécution à la date limite ou le plus proche possible,

l'objectif recherché est alors de minimiser le nombre de tâches en retard. Les auteurs appliquent alors plusieurs règles de priorité en essayant de coupler les techniques pour analyser quelles sont les meilleures heuristiques en cas réel lors de simulations.

Les études sur les tâches-couplées sont souvent motivées par des problèmes de radar comme nous avons pu le voir. Le point négatif est que la plupart des problèmes existants sont $\mathcal{NP}$-complets et les heuristiques données ont souvent un ratio de performance supérieur à 2. Les travaux d'Orman et Potts nous ont permis d'observer les **cas critiques** se trouvant à la frontière entre la polynomialité et la $\mathcal{NP}$-complétude sur la Figure 3.5, lorsque le paramètre $\boldsymbol{L_i}$ est fixé à une constante, la plupart des problèmes sont polynomiaux. Dans la suite nous allons chercher à nous positionner par rapport à ces résultats pour le problème de la torpille en prenant en compte l'introduction du graphe de compatibilité.

3.3.3 Positionnement de notre problème par rapport aux travaux existants

Notre volonté est de **mesurer l'impact** de la **contrainte d'incompatibilité** (resp. de précédence avec des **tâches de traitement**) sur les problèmes d'ordonnancement avec tâches-couplées. Pour cela, nous utilisons comme base théorique les résultats d'Orman et Potts [57] représentés à la Figure 3.5. Remarquons que leurs études portent sur des problèmes en présence de tâches-couplées ayant aucune autre contrainte, leurs travaux peuvent donc être associés aux mêmes problèmes en présence d'un **graphe complet de compatibilité** pour les tâches-couplées. Pour chaque problème des trois treillis représentant la complexité globale, nous ajoutons soient les tâches de traitement, soit le graphe de compatibilité puis les deux en même temps. Pour chaque cas nous étudierons la complexité des problèmes et chercherons comme Orman et Potts les cas critiques entre la polynomialité et la $\mathcal{NP}$-complétude de ces problèmes. Pour chacun nous donnerons des heuristiques générales applicables à tout problème et pour certains cas des heuristiques spécifiques donnant de meilleures bornes d'approximation.

L'ajout des deux contraintes de compatibilité et de précédence nécessite une approche différente pour l'étude de la complexité et de l'approximation de ces problèmes. Le graphe de compatiblité représentant les tâches d'acquisition et le meilleur ordonnancement étant souvent de chevaucher des tâches d'acquisition les unes avec les autres, notre étude portera sur les recouvrements de sommets dans un graphe quelconque, où les cliques donneront des gros blocs de tâches à exécuter, alors que des chaînes créeront beaucoup plus de temps d'inactivité selon la taille du paramètre L_i.

La section suivante présente les principaux résultats de la théorie des graphes sur les recouvrements de sommets d'un graphe quelconque ou particulier.

3.3.4 État de l'art sur les différents recouvrements des sommets d'un graphe par des arêtes

Les problèmes de recouvrements des sommets d'un graphe représentent une grande partie des problèmes de base de la **théorie des graphes**. Ces problèmes sont étudiés depuis plus d'un demi siècle et l'ensemble des travaux sur ce type de problèmes existant dans la littérature est plus que conséquent. Parmi tous les problèmes de recouvrement possibles, nous ne nous intéresserons qu'au recouvrement de sommets par des sous-graphes étant soit des cliques, soit des chaînes. Afin de limiter notre état de l'art sur ces problèmes de recouvrements, nous présenterons quatre problèmes bien connu de la littérature que nous utiliserons le plus pour notre problème :

- PARTITION EN CLIQUES (Problème 2.2.3)
- PARTITION EN SOUS GRAPHES ISOMORPHES (Problème 2.2.4)

– PARTITION EN CHAÎNES (Problème 2.2.6)
– PARTITION EN K-CHAÎNES (Problème 2.2.7)

PARTITION EN CLIQUES

Le problème de PARTITION EN CLIQUES est un problème de décision nommé ainsi par Garey et Johnson dans leur livre [31], mais nommé initialement VERTEX COVER par Karp [42] dans sa preuve de $\mathcal{NP}$-complétude pour des graphes généraux. Les différents travaux réalisés sur ce problème comme d'autres en théorie des graphes sont souvent sur une étude théorique de la complexité de problèmes de décision ayant des classes de graphes particuliers et des sommets de degré fixé.

Pour le problème de PARTITION EN CLIQUES, Garey et Johnson [31] montrent la $\mathcal{NP}$-complétude dans le cas des graphes K_4-libres[5]. Dans le cas des graphes ayant des sommets de degré maximum égal à 4, ils montrent avec Stockmeyer [33] la $\mathcal{NP}$-complétude, et Hunt III et al. [39] montrent le même résultat si ces graphes sont également planaires. D'autres résultats ont été donnés sur ce problème, notons que PARTITION EN CLIQUES est équivalent au problème K-COLORIABLE qui consiste à savoir si un graphe est coloriable en au plus K couleurs (pour plus d'information sur ces deux problèmes voir [7, 17, 33, 38]).

PARTITION EN SOUS GRAPHES ISOMORPHES

Le problème de PARTITION EN SOUS GRAPHES ISOMORPHES d'un graphe G est souvent appelé H-MATCHING dans la littérature. Kirkpatrick et Hell montrent dans [43] la $\mathcal{NP}$-complétude de ce problème pour n'importe quel H fixé $|H| \geq 3$. Le cas où H et G sont des graphes planaires reste ouvert, mais toutefois Bergman et al. [9] donnent une preuve de $\mathcal{NP}$-complétude pour le cas où H est un graphe planaire extérieur connexe[6] tel que $H \geq 4$, et donnent également un algorithme polynomial si H est triangulé tel que $H \geq 4$.

Enfin pour finir avec ce survol des résultats sur le problème PARTITION EN SOUS GRAPHES ISOMORPHES, notons que le problème est $\mathcal{NP}$-complet lorsque H est une clique. Et ajoutant également que le problème PARTITION EN TRIANGLES est un cas particulier de PARTITION EN SOUS GRAPHES ISOMORPHES lorsque H est une clique de taille 3.

PARTITION EN CHAÎNES

Le problème de PARTITION EN CHAÎNES qui consiste à déterminer le nombre minimum de chaînes dans une partition en chaînes, est $\mathcal{NP}$-complet. En effet si le nombre de chaînes est égal à un, ce problème revient à trouver une chaîne Hamiltonienne. Ainsi, le problème PARTITION EN CHAÎNES est $\mathcal{NP}$-complet sur toutes les classes de graphes où le problème de décision CHAÎNE HAMILTONIENNE 2.2.5 est $\mathcal{NP}$-complet comme par exemple les graphes planaires, biparties et triangulés (voir [34]).

D'un autre côté, il a un grand nombre de classes spéciales de graphes pour lesquelles le problème PARTITION EN CHAÎNES est résoluble en temps polynomial. Pour ce qui est des graphes intervalles et circulaires, Bonuccelli et Bovet [13] et Arikati et Pandu [3] donnent des algorithmes en temps linéaire pour résoudre le problème, et il en est de même pour Goodman et Hedetniemi [35] et Misra et Tarjan [53] pour les arbres. Skupien [63] donne un algorithme polynomial pour les forêts, Chang et Kuo [19] et Lin et al. [51] donnent un algorithme en temps

[5] Un K_4-libre est l'appellation standard en théorie des graphes pour nommer un graphe ne contenant aucune clique de taille 4.

[6] Un graphe planaire extérieur (appelé «outerplanar graph» en anglais) est un graphe planaire dont les régions internes créées par les arêtes du graphe sont toutes adjacentes à la région extérieure entourant le graphe.

linéaire pour les co-graphes, et il en est de même pour Srikant et al. [65] (resp. Yan et Chang [70]) pour les graphes de permutation biparti (resp. les graphes blocs).

Le problème de PARTITION EN CHAÎNES trouve ses applications dans la représentation de base de données, les réseaux, les protocoles d'anneaux, l'optimisation de code, etc.

PARTITION EN K-CHAÎNES

Le problème PARTITION EN K-CHAÎNES est une généralisation du problème précédent où toutes les chaînes doivent être de longueur inférieure à un entier K fixé. Il est facile de voir que le cas où $K = 2$ est équivalent à trouver un couplage de taille maximum dans le graphe. Pour ce qui est de $K = 3$, le problème devient $\mathcal{NP}$-complet pour des graphes quelconques [31]. Steiner [66, 67] montre la $\mathcal{NP}$-complétude du problème pour les co-graphes et les graphes triangulés bipartis si K est considéré comme une donnée du problème, et montre également la $\mathcal{NP}$-complétude pour les graphes de comparabilité même si $K = 3$.

Il y a peu de classes de graphes pour lesquelles le problème est résoluble en temps polynomial. Yan et al. [71], et Masuyama et Ibaraki [52] donnent des algorithmes en temps linéaire pour les arbres, Steiner [66] montre que le problème pour les co-graphes devient polynomial si K est fixé, et dans un papier plus récent [67] il propose un algorithme polynomial en temps quadratique pour les graphes de permutation biparti. Enfin Jin et Li [41] développent un algorithme polynomial efficace en temps quadratique pour résoudre le problème pour des cactis.

Le problème PARTITION EN K-CHAÎNES trouve ses applications sur des problèmes de diffusion d'information par ondes ou sur les réseaux informatiques, ou encore sur des problèmes de tournée de véhicules.

3.4 Conclusion

Ce chapitre se présente en deux parties, dans la première nous avons proposé une modélisation du problème pratique, où certaines contraintes réelles ont été écartées pour modéliser un problème théorique. Le problème porte sur deux ensembles de tâches, d'acquisition (similaire aux tâches-couplées introduites par Shapiro [61]) et de traitement, ayant des contraintes de précédence et d'incompatibilité pour les tâches d'acquisition.

La deuxième partie du chapitre a été consacrée à donner un rapide état de l'art pour l'ordonnancement classique sur mono processeur, puis à une présentation des résultats pour l'ordonnancement avec tâches-couplées. Ces derniers résultats nous ont permis de positionner notre problème par rapport aux travaux existants, et ainsi d'organiser notre objectif principal dans la thèse : **mesurer l'impact de la contrainte d'incompatibilité (resp. de l'introduction des tâches de traitement) sur les problèmes d'ordonnancement avec tâches-couplées**. La présence d'un graphe de compatibilité conduit à recenser des problèmes de la théorie des graphes portant sur la recherche de structures ou propriétés particulières sur des sous-graphes.

Ce chapitre clos la première partie sur la présentation générale du manuscrit et du problème étudié. Nous allons maintenant étudier la complexité des différents problèmes rencontrés en faisant varier les paramètres fondamentaux. Cette étude nous permettra de mieux évaluer l'impact de l'introduction du graphe de compatibilité et des tâches de traitement.

Deuxième partie

Étude de la complexité

Chapitre 4

Prise en compte de la contrainte d'incompatibilité

4.1 Introduction

Ce chapitre porte sur l'étude des problèmes d'ordonnancement avec tâches-couplées en présence d'un **graphe de compatibilité n'ayant aucune structure particulière**. En se basant sur les résultats d'Orman et Potts [57], qui donnent à travers trois treillis une visualisation globale de la complexité des problèmes d'ordonnancement avec tâches-couplées en présence d'un graphe de compatibilité et ayant pour particularité que chaque sommet est connecté à tous les autres (voir la Figure 3.5), notre démarche scientifique consiste à réaliser le même type d'étude lorsque la contrainte d'incompatibilité est introduite, c'est-à-dire en présence d'un graphe de compatibilité de structure quelconque.

En reprenant la notation de la Figure 3.5, nous présentons sur la Figure 4.1 la vision globale de la complexité des mêmes problèmes d'ordonnancement avec la présence d'un graphe de compatibilité ayant une structure quelconque.

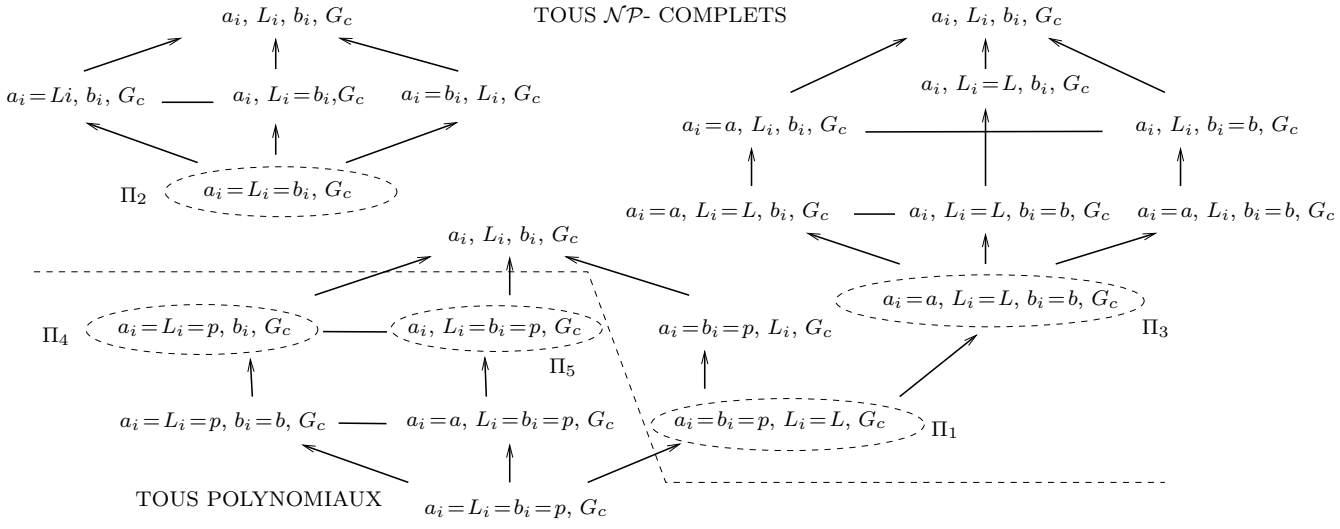


Fig. 4.1 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition en présence de contrainte d'incompatibilité

La prise en compte de la contrainte d'incompatibilité rendant les problèmes plus complexes qu'ils ne l'étaient, nous obtenons des résultats immédiats de complexité :

Remarque 4.1.1 :
Les problèmes qui étaient $\mathcal{NP}$-complets avec un graphe de compatibilité complet sont des cas particuliers et restent trivialement $\mathcal{NP}$-complets avec un graphe quelconque (voir le Lemme 2.1.3 page 13 de la Section 2.1.4).

Le principal changement de complexité vient des problèmes notés Π_1, Π_3 sur la Figure 4.1 page 41 qui deviennent $\mathcal{NP}$-complets avec la prise en compte de la contrainte d'incompatibilité. Nous montrerons que la $\mathcal{NP}$-complétude de Π_1 entraîne celle de Π_3.

En considérant la hiérarchie entre nos problèmes, il nous suffit d'étudier certains cas particuliers pour obtenir la complexité de tous les autres problèmes. Nous focaliserons donc notre étude sur les problèmes notés Π_1, Π_2, Π_3, Π_4 et Π_5 (voir p. 41) qui se situent à la frontière de la $\mathcal{NP}$-complétude et la polynomialité par rapport au schéma de la Figure 4.1 (la complexité de Π_2 étant déjà donné, nous nous intéresserons uniquement à son approximation). Pour ces problèmes là, nous fixerons certains paramètres afin de mieux évaluer l'influence du graphe de compatibilité sur le passage de la polynomialité à la $\mathcal{NP}$-complétude.

4.1.1 Présentation du chapitre

Dans la première section nous donnerons les preuves de $\mathcal{NP}$-complétude et de polynomialité des cinq problèmes qui nous intéressent, suivies d'une analyse plus poussée selon les valeurs des paramètres fondamentaux. Puis dans la deuxième section, nous donnerons plusieurs heuristiques pour les différents problèmes étudiés, en prenant compte à nouveau des valeurs des paramètres selon les cas.

4.2 Classification de problèmes par la complexité

Nous allons montrer dans un premier temps la $\mathcal{NP}$-complétude de deux problèmes d'ordonnancement suivants :

- $\Pi_1 : 1|a_i = b_i = p, L_i = L, G_c|C_{max}$ (Section 4.2.1 page 42)
- $\Pi_3 : 1|a_i = a, L_i = L, b_i = b, G_c|C_{max}$ (Section 4.2.2 page 45)

Dans un deuxième temps nous nous intéresserons à la polynomialité des problèmes d'ordonnancement suivants :

- $\Pi_4 : 1|a_i = L_i = p, b_i, G_c|C_{max}$ (Section 4.2.3 page 47)
- $\Pi_5 : 1|a_i, L_i = b_i = p, G_c|C_{max}$ (Section 4.2.4 page 50)

Une fois que nous aurons prouvé la polynomialité de ces deux problèmes, nous en déduirons la polynomialité de tous les problèmes plus spécifiques. Intuitivement, les algorithmes appliqués pour résoudre ces problèmes sont tous basés sur la recherche d'un couplage de taille maximum du fait de la structure particulière des tâches d'acquisition, $a_i = L_i = p$ et/ou $L_i = b_i = p$ dans tous les cas polynomiaux.

4.2.1 Étude du problème $\Pi_1 : 1|a_i = b_i = p, L_i = L, G_c|C_{max}$, avec $L, p \in \mathbb{N}^*$

D'après Orman et Potts, le problème $1|a_i = b_i = p, L_i = L, G_c \textit{ complet}|C_{max}$ est polynomial. Nous allons aborder le problème du point de vue de la complexité en faisant varier le paramètre L. Nous pouvons exhiber trois cas distincts, les deux premiers sont polynomiaux, et le dernier $\mathcal{NP}$-complet :

Lemme 4.2.1 :
Il existe un algorithme en temps polynomial permettant de résoudre le problème Π_1 *avec* $0 < L < p$.

Preuve :
Lorsque $0 < L < p$, aucune tâche ne peut chevaucher son exécution avec celle d'une autre tâche. Il est donc évident que l'ordonnancement optimal revient à exécuter de manière gloutonne les tâches une à côté de l'autre le plus tôt possible de telle sorte que pour deux tâches consécutives A_i et A_j, la date $\sigma(A_j)$ de début d'exécution de la seconde tâche A_j soit égale à la date $C(A_i)$ de complétion de la première tâche A_i. Cet algorithme admet une complexité en temps linéaire.

□

Lemme 4.2.2 :
Il existe un algorithme en temps polynomial permettant de résoudre le problème Π_1 *avec* $p \leq L < 2p$.

Preuve :
Le slot créé par chaque tâche d'acquisition ne permet pas de faire chevaucher plus de deux tâches en même temps. Ainsi tout ordonnancement de Π_1 peut être associé à un couplage des sommets de G_c, les tâches associées aux sommets couverts par les arêtes du couplages sont exécutées deux à deux (créant ainsi des blocs avec un temps d'inactivité forcément inférieur à p), pour deux tâches A_i et A_j nous avons $\sigma(A_j) = \sigma(A_i) + a_i$. Après avoir ordonnancé les tâches correspondantes au couplage, nous exécutons le reste des tâches qui n'appartiennent pas au couplage de la même manière qu'au premier cas. La longueur de l'ordonnancement va donc dépendre de la taille du couplage, et ainsi la recherche d'un couplage de cardinalité maximum dans G_c permet d'obtenir un ordonnancement optimal. La recherche d'un couplage maximum dans un graphe général étant de complexité $O(n^3)$ en utilisant l'algorithme de Gabow [29], le cas $p \leq L < 2p$ est bien polynomial.

□

Lorsque $L \geq 2p$, nous pouvons désormais faire chevaucher l'exécution de plus de deux tâches d'acquisition, ce qui nous amène à rechercher des cliques dans le graphe de compatibilité pour limiter le temps d'inactivité sur le processeur. Nous allons montrer que la relaxation des contraintes sur le graphe de compatibilité entraîne la $\mathcal{NP}$-complétude.

Nous prenons un cas particulier de Π_1 dans lequel L_i admet une longueur fixe deux fois plus grande que a_i et b_i. Ce problème sera noté $\Pi'_1 : 1|a_i = b_i = p, L_i = 2p, G_c|C_{max}$. Si Π'_1 est $\mathcal{NP}$-complet alors le cas général Π_1 sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3 de la Section 2.1.4.

Théorème 4.2.1 :
Le problème de décider si une instance de notre problème Π'_1 *possède un ordonnancement de longueur* $\beta = \sum_{i=1}^{n}(a_i + b_i) = 2np$ *est* $\mathcal{NP}$*-complet.*

Preuve :
Tout d'abord, notons qu'il est trivial de voir que le problème de décision Π'_1 est dans $\mathcal{NP}$. Maintenant, afin de prouver que ce problème est $\mathcal{NP}$-complet, une réduction polynomiale va être réalisée à partir du problème de décision PARTITION EN TRIANGLES (Problème 2.2.1) vers Π'_1.

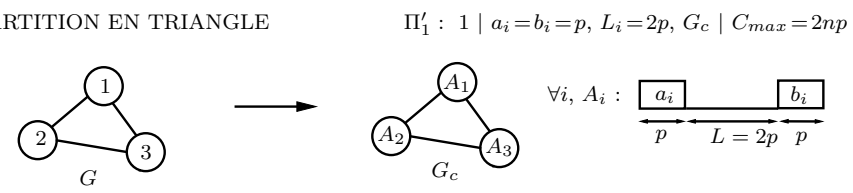


Fig. 4.2 – Exemple de transformation polynomiale

Soit une instance I^* de PARTITION EN TRIANGLES, construisons en temps polynomial une instance I de $1 \mid a_i = b_i = p,\ L_i = 2p,\ G_c \mid C_{max} = 2np$ de la manière suivante :

Soit $G = (V, E)$ le graphe de l'instance I^*. $G_c = (\mathcal{A}, E_c)$ est construit de la manière suivante (voir illustration Figure 4.2) :

- $\forall i \in V$, où $|V| = 3q$, une tâche d'acquisition A_i est introduite dans $\mathcal{A}$, composée de deux sous-tâches a_i et b_i de durée d'exécution $a_i = b_i = p$ et d'un temps d'attente incompressible et indilatable entre ces deux sous-tâches de longueur $L_i = 2p$.
- Pour chaque arête $e = (i, j) \in E$ dans G, une arête $e_c = (A_i, A_j) \in E_c$ est ajoutée dans G_c, on a une relation de non exclusivité entre les deux tâches A_i et A_j.

Cette transformation est clairement calculable en temps polynomial.

Maintenant que l'instance I est construite, montrons que l'existence d'un recouvrement par triangles des sommets de G implique l'existence d'un ordonnancement optimal sans temps d'inactivité, et réciproquement :

⇒ Supposons qu'il existe un recouvrement en triangles des sommets du graphe G. Montrons alors qu'il existe un ordonnancement sans temps d'inactivité en $2np$ unités de temps ($2np$ représentant la somme des temps d'exécution). Pour cela, il suffit de regrouper les tâches d'acquisitions A_i trois par trois dans G_c selon le recouvrement en triangles trouvé dans G. L'exécution de ces groupes de tâches forme un ordonnancement sans temps d'inactivité (voir illustration Figure 4.3), et donc nous avons un ordonnancement réalisable en $2np$ unités de temps.

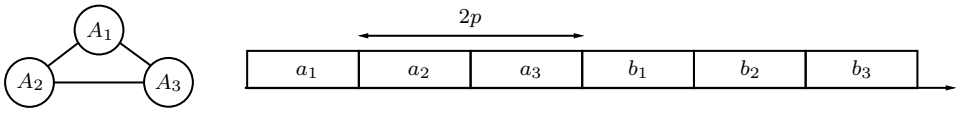


Fig. 4.3 – Illustration de trois tâches d'acquisition formant un bloc, $\sigma(a_3) = C(a_2) = C(a_1) + a_2$

⇐ Réciproquement, s'il existe un ordonnancement en $2np$ unités de temps, montrons alors que les sommets de G peuvent être recouverts par des triangles.

Il est clair que si $C_{max} = 2np$, il n'y a aucun temps d'inactivité sur le processeur. Ceci entraîne que chaque slot d'inactivité de longueur $L_i = 2p$ sera forcément rempli. Or il faut trois tâches d'acquisition exécutées les unes dans les autres pour avoir un bloc de trois tâches sans temps d'inactivité. Donc, en ayant exactement q blocs, nous avons bien un ordonnancement sans temps d'inactivité. Et puisque trois tâches d'acquisition exécutées les unes dans les autres sont forcément compatibles dans G_c, il existe un recouvrement par triangles des sommets de G_c et des sommets de G par construction.

Nous avons donc PARTITION EN TRIANGLES $\leq_T \Pi'_1$, et nous savons que PARTITION EN TRIANGLES (Problème 2.2.1) est $\mathcal{NP}$-complet. Ainsi, d'après le Lemme 2.1.2 de la Section 2.1.4, nous pouvons conclure que le problème Π'_1 est $\mathcal{NP}$-complet.

□

Remarque 4.2.1 :
Au vu de la preuve de $\mathcal{NP}$-complétude, nous remarquons que pour $L = kp$, avec $k \geq 2$, l'existence d'un ordonnancement sans trou revient à trouver une partition des sommets de G_c par des cliques disjointes de taille $k+1$ (qui est équivalent au problème $\mathcal{NP}$-complet PARTITION EN SOUS-GRAPHES ISOMORPHES À H, où H est une clique de taille $(k+1)$).

L'étude de l'approximation du problème Π_1 se trouve à la Section 4.3.1 page 51.

4.2.2 Étude du problème $\Pi_3 : 1|a_i = a, L_i = L, b_i = b, G_c|C_{max}$, avec $a, b, L \in \mathbb{N}^*$

D'après les résultats obtenus par Orman et Potts [57] (voir Figure 3.5), le problème $1|a_i = a, b_i = b, L_i = L, G_c$ complet $|C_{max}$ est toujours ouvert. En faisant varier les valeurs des paramètres fondamentaux a_i et b_i, nous pouvons proposer la remarque suivante :

Remarque 4.2.2 :
Le problème $\Pi_3 : 1|a_i = a, b_i = b, L_i = L, G_c|C_{max}$ est une généralisation du problème $\Pi_1 : 1|a_i = b_i = p, L_i = L, G_c|C_{max}$. En effet, en posant $a = b = p$ sur une instance de Π_3 nous obtenons une instance de Π_1.

Théorème 4.2.2 :
Le problème de décision Π_3 est $\mathcal{NP}$-complet.

Preuve :
La Remarque 4.2.2 implique la $\mathcal{NP}$-complétude du problème Π_3 d'après le Lemme 2.1.3 de la Section 2.1.4.

□

Nous venons de montrer que le problème $\Pi_3 : 1|a_i = a, b_i = b, L_i = L, G_c|C_{max}$ était $\mathcal{NP}$-complet dans le cas général, nous allons approfondir l'étude de la complexité de ce cas particulier en fonction de a et b.

Nous allons établir la complexité de Π_3 selon les valeurs de a, b et L. D'un côté, nous montrons que le problème est polynomial pour tout $L < a + b$, de l'autre le problème devient $\mathcal{NP}$-complet pour $L \geq a + b$.

Lemme 4.2.3 :
Il existe un algorithme en temps polynomial permettant de résoudre le problème Π_3 lorsque $L < a + b$.

Preuve :
Il est évident de voir que chercher un couplage de taille maximum dans le graphe G_c garantit une solution optimale. En effet, durant le slot L d'une tâche d'acquisition A_i, nous pouvons exécuter au plus une sous-tâche a_j ou b_j, pour $i \neq j$. Puisque les tâches d'acquisition sont toutes équivalentes, l'algorithme consistant à chercher un couplage de cardinalité maximum dans le graphe G_c puis à ordonnancer les tâches selon le couplage donne une solution optimale au problème. Pour toute arête (A_i, A_j) du couplage, $\sigma(A_j) = \sigma(A_j) + a_i$, et pour les sommets isolés, les tâches sont exécutées consécutivement.

Donc le problème $1|a_i = a, b_i = b, L_i = L < a + b, G_c|C_{max}$ admet un algorithme en temps polynomial de complexité $O(n^3)$ en utilisant l'algorithme de Gabow [29].

□

Regardons maintenant le cas $L \geq a + b$. Remarquons tout d'abord que le cas $a = b$ est équivalent au problème $\Pi_1' : 1|a_i = b_i = p, L \geq 2p, G_c|C_{max}$ étudié dans la Section 4.2.1. Nous allons montrer que le problème Π_3 est $\mathcal{NP}$-complet lorsque $L = a + b$, et nous en déduirons ainsi la $\mathcal{NP}$-complétude pour $L \geq a + b$. Le problème que nous allons étudier dans la suite est le cas $L = a + b$ avec $a > b$, les résultats que nous présentons ici peuvent être symétriquement étendus aux instances où $b > a$.

Considérons un ordonnancement valide σ d'une instance $(\mathcal{A}, G_c)$ de Π_3 avec $L = a + b$ et $a > b$, composée d'un ensemble de tâches-couplées $\mathcal{A}$ et d'un graphe de compatibilité G_c. Avant de donner un résultat de complexité, nous allons analyser les différentes façons d'exécuter les tâches d'acquisition. Pour une tâche A_i donnée, au plus deux autres sous-tâches peuvent s'exécuter entre la date $C(a_i)$ de complétion de a_i et la date $\sigma(b_i)$ de début d'exécution de b_i. Dans ce cas, l'ordonnancement de longueur minimum consiste à exécuter une sous-tâche b_j et une sous-tâche a_k durant le temps d'inactivité L_i, avec $i \neq j \neq k$ tels que $\sigma(b_j) = \sigma(a_i) + a$ et $\sigma(a_k) = \sigma(a_i) + a + b$. La Figure 4.4 illustre une telle configuration.

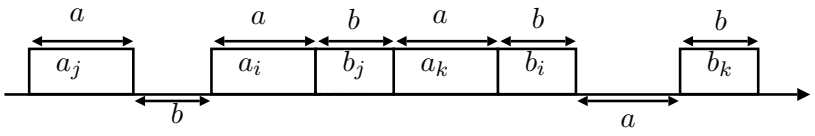
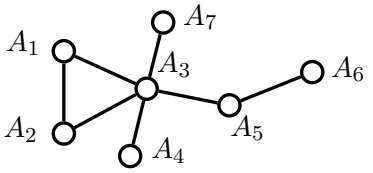
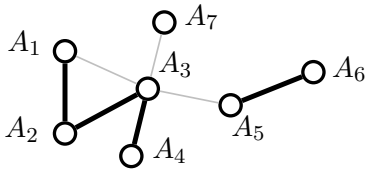


Fig. 4.4 – Au plus deux sous-tâches peuvent être ordonnancées entre a_i et b_i

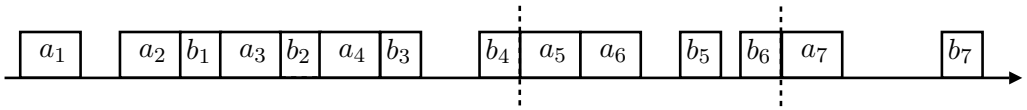
Nous pouvons en déduire que n'importe quel ordonnancement valide σ peut être vu comme une partition $\{V_1, V_2, \ldots, V_k\}$ de $\mathcal{A}$, telle que pour tout V_i il existe une chaîne C_i passant par chaque sommet de V_i (les sommets isolés sont considérés comme des chaînes de longueur 0). Clairement, $\{C_1, C_2, \ldots, C_k\}$ est une partition de G_c en chaînes disjointes. La Figure 4.5 montre une instance de Π_3 (Figure 4.5(a)), un ordonnancement valide (Figure 4.5(c)), qui n'est pas forcément optimal, et la partition de G_c en chaînes disjointes correspondantes (Figure 4.5(b)).



(a) Un graphe G_c avec 7 tâches



(b) Une partition disjointe



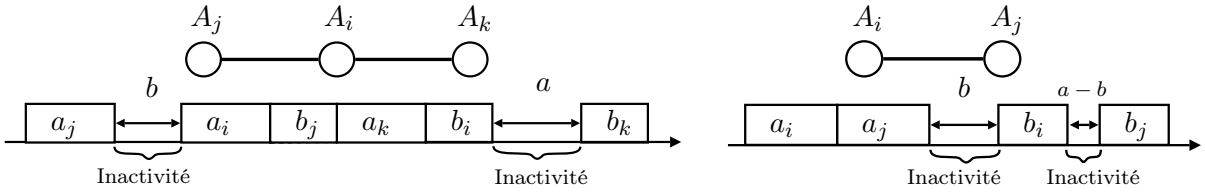
(c) L'ordonnancement correspondant à la partition

Fig. 4.5 – Relation entre un ordonnancement et une partition en chaînes disjointes

Pour un ordonnancement σ donné, nous analysons la relation entre la longueur de l'ordonnancement et la partition en chaînes disjointes correspondantes $\{C_1, C_2, \ldots C_k\}$. Clairement, nous avons $C_{max} = T_{seq} + T_{idle}$[7], où $T_{seq} = n(a + b)$ et T_{idle} dépend de la partition obtenue.

[7]Rappelons que T_{seq} représente la somme des temps séquentiel de toutes les tâches et T_{idle} la somme des temps d'inactivité comme défini à la page 10.

Nous allons avoir deux types de chaînes, pour n'importe quelle chaîne de longueur strictement plus grande que un, le temps d'inactivité de cette chaîne augmente le T_{idle} de $(a + b)$ (voir Figure 4.6(a)). Remarquons que pour une chaîne de longueur 0 (correspondant à une tâche isolée), le temps d'inactivité ajouté est également égal à $(a + b)$. Et enfin pour toute chaîne de longueur un (correspondant à une arête), le temps d'inactivité de l'arête est seulement égal à a puisque les deux tâches correspondantes peuvent être imbriquées comme sur la Figure 4.6(b).



(a) Chaînes de longueur supérieure à 2 avec $T_{idle} = a + b$ (b) Chaînes de longueur 1 avec $T_{idle} = a$

Fig. 4.6 – Impact de la longueur des chaînes sur le temps d'inactivité

Cette analyse entraîne un théorème immédiat sur la complexité de notre cas :

Théorème 4.2.3 :
Le problème de décider s'il existe un ordonnancement en $(n + 1)(a + b)$ *unités de temps du problème* Π_3 *avec* $L = a + b$*, avec* $a > b$*, est* $\mathcal{NP}$*-complet.*

Preuve :
La preuve est basée sur une réduction en temps polynomial du problème CHAÎNE HAMILTONIENNE (problème 2.2.5) vers notre problème. Si $C_{max} = (n+1)(a+b)$, le fait d'avoir le temps séquentiel égal à $T_{seq} = n(a+b)$ entraîne que le temps d'inactivité est égal $T_{idle} = a+b$. Le seul recouvrement possible dans G_c pour obtenir ce temps d'inactivité est de recouvrir par une seule chaîne (sinon $T_{idle} = 2a$). Et donc le graphe contient une chaîne hamiltonienne. Réciproquement, si G_c contient une chaîne hamiltonienne, nous pouvons en déduire un ordonnancement avec $C_{max} = (n+1)(a+b)$, où $T_{seq} = n(a+b)$, et T_{idle} doit être égal à $(a+b)$, ce qui est possible si et seulement si nous pouvons ordonnancer toutes les tâches en un seul bloc.

□

Remarque 4.2.3 :
À partir du théorème 4.2.3, nous en déduisons trivialement la $\mathcal{NP}$*-complétude du problème* Π_3 *pour* $L \geq a + b$*.*

L'étude de l'approximation du problème Π_3 se trouve à la Section 4.3.3 page 59.

4.2.3 Étude du problème $\Pi_4 : 1|a_i = L_i = p, b_i, G_c|C_{max}$, avec $p \in \mathbb{N}^*$

Ce problème est composé de n tâches d'acquisition toutes sur le même modèle $a_i = L_i = p, b_i$. La première sous-tâche et le temps d'inactivité sont fixés à la même constante p, $p \in \mathbb{N}^*$. La seconde tâche peut prendre n'importe quelle valeur.

L'ensemble de ces tâches d'acquisition contient deux sous-ensembles de tâches : le premier sous-ensemble noté K est composé de toutes les tâches d'acquisition A_i telles que $b_i \leq p$, le second sous-ensemble noté S est composé par toutes les autres tâches. Deux tâches S_i et S_j dans S ne peuvent pas être emboîtées l'une dans l'autre, donc l'arête $\{i, j\} \notin G_c$ et nous

enlevons automatiquement ces arêtes. Pour cette section, nous dirons que G_c est un graphe complet lorsque K formera une clique, S un stable et $\forall x \in K, \forall y \in S$, nous avons $\{x, y\} \in G_c$.

Théorème 4.2.4 :
Le problème d'ordonnancement $\Pi_4 : 1|\ a_i = L_i = p, b_i, G_c\ |C_{max}$ *est polynomial.*

Preuve :
Nous allons étudier les différentes valeurs de b_i pour chaque tâche A_i. Les seuls choix d'ordonnancement que nous ayons sont soit d'exécuter deux tâches l'une dans l'autre, soit d'exécuter une tâche toute seule. En pondérant chaque arête du graphe par le temps séquentiel du chevauchement des deux tâches extrémités de l'arête, notre problème aura une solution en cherchant un couplage qui minimise le poids des arêtes du couplage et des sommets isolés.

À ces fins, nous allons chercher un problème similaire qui est connu pour être résolu en temps polynomial. Ce problème, qui est équivalent à notre problème via une transformation polynomiale, consiste à chercher un couplage parfait de poids minimum (Problème 2.2.2) qui se fait en temps polynomial [25]. Afin d'avoir un graphe avec un nombre de sommets pair et tel que trouver un couplage parfait soit possible, nous donnons la construction polynomiale suivante :

1. Soit $\mathcal{I}_1$ une instance de notre problème avec un graphe de compatibilité $G_c = (V_c, E_c)$, et $\mathcal{I}_2$ une instance du problème du couplage parfait de poids minimum dans un graphe construit à partir de $\mathcal{I}_1$. Considérons un graphe H_c constitué de deux copies de G_c notée $G'_c = (V'_c, E'_c)$ et $G''_c = (V''_c, E''_c)$. Le sommet correspondant à A_i est noté A'_i dans G'_c et A''_i dans G''_c. De plus, $\forall i = 1, \ldots, n$, une arête est ajoutée entre A'_i et A''_i. Nous avons $H_c = G'_c \cup G''_c = (V'_c \cup V''_c, E'_c \cup E''_c)$, avec $|V'_c \cup V''_c|$ de taille paire.
2. Chaque arête $\{A'_i, A'_j\}$ (resp. $\{A''_i, A''_j\}$), où $b_i > p$ ou $b_j > p$, est pondérée par $\frac{3p+max\{b_i,b_j\}}{2}$. Cette valeur représente la moitié du temps d'exécution utilisé lors de l'ordonnancement des deux tâches-couplées dont la deuxième est une tâche de S.
3. Chaque arête $\{A'_i, A'_j\}$ (resp. $\{A''_i, A''_j\}$), où $b_i \leq p$ et $b_j \leq p$, est pondérée par $\frac{3p+min\{b_i,b_j\}}{2}$. Cette valeur représente la moitié du temps d'exécution utilisé lors de l'ordonnancement des deux tâches-couplées appartenant à K. La seconde tâche exécutée sera celle dont le b_i est le plus petit.
4. Chaque arête $\{A'_i, A''_i\}$ est pondérée par $2p+b_i$. Cette valeur représente le temps d'exécution utilisé lors de l'ordonnancement d'une tâche isolée.

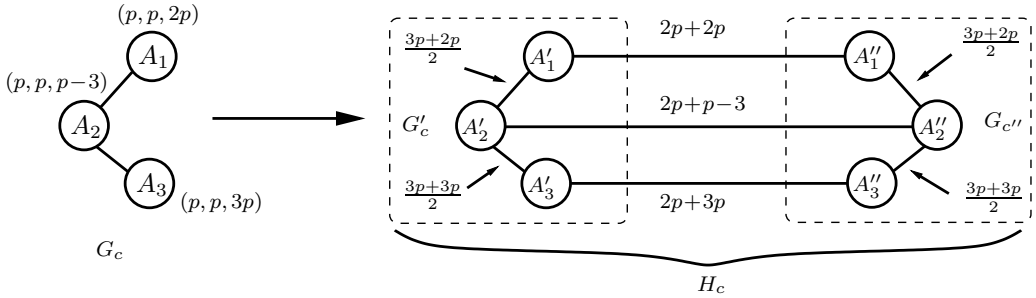


Fig. 4.7 – Exemple de la transformation

Dans le but de proposer un algorithme polynomial résolvant le problème Π_4, nous allons prouver dans un premier temps la proposition suivante.

Proposition 4.2.1 :
A un couplage parfait de poids minimum C, on peut associer un ordonnancement de durée minimum C et réciproquement.

Preuve :
D'après la construction donnée précédemment, le poids de chaque arête $e = (A'_i, A'_j) \in E'_c \cup E'_c$ (resp. $e = (A''_i, A''_j) \in E''_c \cup E''_c$), avec $i \neq j$, correspond à la moitié de la longueur de l'ordonnancement sur le processeur pour les tâches d'acquisition A'_i et A'_j (A''_i et A''_j) si elles se chevauchent. Ce chevauchement peut être représenté par un bloc. Le poids de chaque arête $e = (A'_i, A''_i) \in E'_c \cup E''_c$ correspond à la longueur de l'ordonnancement sur le processeur d'une simple tâche d'acquisition.

Par construction H_c contient un nombre pair de sommets, et du fait que chaque sommet de G'_c est relié à un sommet équivalent dans G''_c, trouver un couplage parfait sur le graphe H_c est possible, ce qui entraîne qu'il existe un ordonnancement tel que chaque tâche n'est exécutée qu'une seule fois. Remarquons que le couplage sur G'_c n'est pas forcément identique à celui sur G''_c, mais ils ont quand même le même poids. Le makespan obtenu est égal à la somme des durées des blocs obtenus et de celles des tâches isolées, et comme chaque bloc a pour durée le poids de l'arête équivalente dans le couplage parfait, nous avons la somme des poids des arêtes du couplage qui est égale à celle des blocs de l'ordonnancement obtenu.

Donc pour un couplage parfait de poids minimum C, nous pouvons associer un ordonnancement de durée minimum C et réciproquement. Ceci finit la preuve de la Proposition 4.2.1.

□

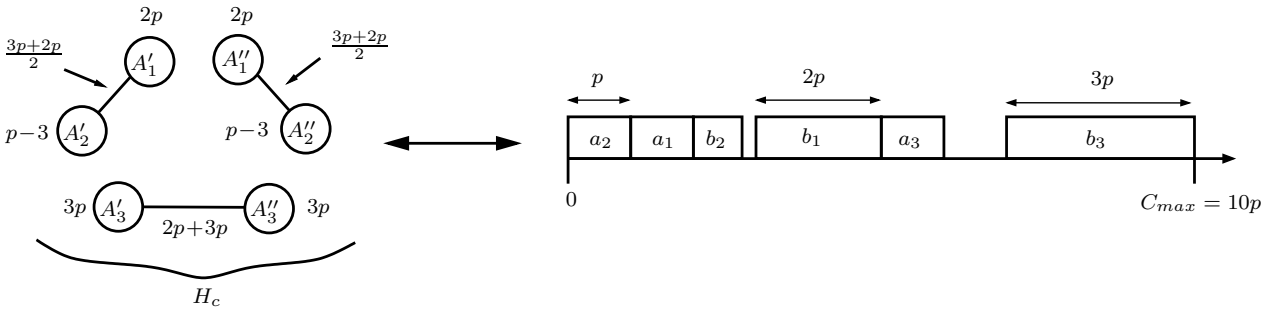


Fig. 4.8 – Exemple de correspondance entre un couplage parfait et un ordonnancement.

Suite de la preuve du Théorème 4.2.4 :
Le proposition 4.2.1 montre le lien entre une solution au problème Π_4 et celle de trouver un couplage parfait de poids minimum dans H_c. Or l'algorithme d'Edmonds permet de trouver un couplage parfait de poids minimum en temps $O(n^2m)$ [25]. Donc le problème d'optimisation Π_4 est polynomial.

□

L'algorithme en temps polynomial donnant une solution optimale au problème $\Pi_4 : 1|\ a_i = L_i = p, b_i, G_c\ |C_{max}$ sera donc décomposé en deux étapes : la première consiste à créer le graphe H_c puis trouver un couplage parfait dans celui-ci, la deuxième étape consiste à exécuter les tâches d'acquisition sur le processeur selon les arêtes du couplage. L'algorithme 4 donne une telle solution avec un temps de complexité de $O(n^2m)$.

Algorithme 4 : Un ordonnancement optimal en temps polynomial

Données : $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, H_c, G_c
Résultat : C_{max}^{opt}
début
- Chercher dans H_c un couplage parfait M minimisant le poids des arêtes du couplage
- Pour chaque arête $e = (A'_i, A'_j) \in H_c$ (resp. $e = (A''_i, A''_j) \in H_c$) du couplage M telle que A'_i et A'_j (resp. A''_i et A''_j) appartiennent au même graphe G'_c (resp. G''_c), les tâches d'acquisition A_i et A_j associées au graphe G_c sont ordonnancées l'une dans l'autre selon le poids de l'arête. Deux cas sont alors possibles, si $p \geq b_i \geq b_j$ alors $\sigma(A_j) = \sigma(A_i) + a_i$, et si $b_i \geq p$ alors $\sigma(A_i) = \sigma(A_j) + a_i$.
- Pour chaque arête $e = (A'_i, A''_i) \in H_c$ du couplage M telle que $A'_i \in G'_c$ et $A''_i \in G''_c$, la tâche d'acquisition A_i associée au graphe G_c est exécutée à la suite de l'ordonnancement.

fin

4.2.4 Étude du problème $\Pi_5 : 1|\ a_i, L_i = b_i = p, G_c|C_{max}$

Ce problème est composé de n tâches d'acquisition toutes sur le même modèle $(a_i, L_i = b_i)$. Chaque tâche d'acquisition est différente des autres, et pour chacune d'elles les deux sous-tâches et le temps d'inactivité ont la même durée d'exécution.

Théorème 4.2.5 :
Le problème d'ordonnancement $\Pi_5 : 1|\ a_i, L_i = b_i = p, G_c\ |C_{max}$ *peut être résolu en temps polynomial.*

Preuve :
Orman et Potts ont montré une équivalence très utile pour l'étude de certains cas dans [57]. Ils démontrent qu'un problème d'ordonnancement défini par $1|a_i, L_i, b_i, G_c\ complet|C_{max}$, avec $i = 1, \ldots, n$, est équivalent à un problème $1|b_i, L_i, a_i, G_c\ complet|C_{max}$. On dit alors que ces deux problèmes sont **symétriques**[8]. Afin d'utiliser cette caractéristique particulière pour étudier la complexité des problèmes, Orman et Potts [57] donnent le théorème suivant sur les problèmes d'ordonnancement avec tâches-couplées .

Théorème 4.2.6 :
Un problème d'ordonnancement avec tâches-couplées en présence d'un graphe de compatibilité complet, dont l'objectif est le makespan, est de même complexité qu'un problème qui lui est symétrique. Ce qui n'est pas le cas pour l'approximation.

Ce théorème nous permet de donner la complexité d'un problème lorsque nous connaissons celle du problème symétrique. Dans la Figure 3.5, nous avons une symétrie entre $1|a_i = L_i = p, b_i|C_{max}$ et $1|a_i, L_i = b_i = p|C_{max}$. En relaxant la contrainte d'incompatibilité, les deux problèmes restent symétriques, donc le problème Π_5 est symétrique au problème Π_4.

D'après le Théorème 4.2.6, nous en déduisons que le problème Π_5 est **polynomial** tout comme l'est le problème Π_4. Nous avons un ordonnancement optimal avec l'Algorithme 1 en changeant b_i par a_i.

□

[8]Deux problèmes sont symétriques si les caractéristiques de a_i (resp. b_i) du premier problème sont les mêmes que celles de b_i (resp. a_i) pour l'autre problème. Par exemple $1|a_i = a, L_i, b_i, G_c\ complet|C_{max}$ et $1|a_i, L_i, b_i = b, G_c\ complet|C_{max}$ sont symétriques, avec $a, b \in \mathbb{N}$.

4.2.5 Vision globale de la complexité

Nous avons montré la $\mathcal{NP}$-complétude des deux problèmes Π_1 et Π_3, et la polynomialité des deux problèmes Π_4 et Π_5 (voir page 41 pour la définition de ces problèmes). Comme nous l'avons indiqué dans l'introduction de ce chapitre, tous les problèmes qui étaient déjà $\mathcal{NP}$-complets avec un graphe de compatibilité complet (voir Figure 3.5 page 33) restent $\mathcal{NP}$-complets lorsque G_c a une structure quelconque (voir Figure 4.1).

Pour ce qui est des problèmes qui étaient polynomiaux avec un graphe de compatibilité complet, la complexité reste la même avec un graphe quelconque. En effet, à partir des résultats de complexité de Π_4 et Π_5 et d'après le Lemme 2.1.3, tous les problèmes plus spécifiques que Π_4 et Π_5 sont également polynomiaux.

Ceci finit les preuves de complexité pour les problèmes d'ordonnancement avec tâches-couplées en présence d'un graphe de compatibilité ayant une structure quelconque. Dans la section suivante, nous allons continuer l'analyse des problèmes Π_1, Π_2 et Π_3 d'un point de vue de l'approximation.

4.3 Étude de l'approximation

Cette section portera sur des études d'approximation focalisées sur les problèmes $\mathcal{NP}$-complets Π_1, Π_2 et Π_3. Après une première étude rapide, nous essayerons d'obtenir de meilleurs bornes selon les différentes valeurs des paramètres, ou encore la topologie du graphe de compatibilité.

4.3.1 Étude d'approximation pour $\Pi_1 : 1|a_i = b_i = p, L_i = L, G_c|C_{max}$

Intéressons nous à l'approximation du problème $\mathcal{NP}$-complet Π_1, l'étude de la complexité de ce problème a été faite à la Section 4.2.1 page 42. Rappelons que nous travaillons sur n tâches d'acquisition, et lorsque $L \geq 2np$ la prise en compte de la contrainte d'incompatibilité entraîne la $\mathcal{NP}$-complétude du problème. La recherche d'une heuristique avec garantie de performance non triviale donnant un ordonnancement le plus proche possible de l'optimal passera forcément par une étude sur le graphe de compatibilité G_c. Nous allons donner deux bornes inférieures, et une borne supérieure obtenue avec un recouvrement par cliques maximales des sommets de G_c.

Borne inférieure pour toute solution optimale

Lemme 4.3.1 *En considérant un couplage maximum M dans G_c de taille m, notre borne inférieure sera $C^{opt}_{max} \geq max\{2np, (n-2m)(L+2p)\}$*

Preuve :
L'ordonnancement optimal est obtenu lorsque nous n'avons aucun temps d'inactivité, c'est à dire lorsque les tâches d'acquisition forment des blocs composés chacun par $\beta = (\frac{L}{p} + 1)$ tâches A_i, où $L = kp$ avec $k \in \mathbb{N}^*$. Ces blocs se traduisent par un recouvrement des sommets de G_c par des cliques de taille β. La borne inférieure de notre étude sera donc :

$$C^{opt}_{max} \geq T_{seq} = 2np \tag{4.1}$$

De plus, en considérant un couplage de taille maximum M dans le graphe de compatibilité de taille m, le nombre de sommets isolés s'élèvent à $(n-2m)$. Dans le pire des cas, l'ordonnancement optimal est forcément supérieur à l'ordonnancement des sommets isolés qui forment un stable

entre eux. Ainsi nous obtenons une deuxième borne inférieure selon un couplage maximum M de taille m :

$$C_{max}^{opt} \geq (n - 2m)(L + 2p) \tag{4.2}$$

Donc selon les valeurs que prendront les paramètres dans notre étude, notre borne inférieure dépendra soit de celle de l'équation 4.1 soit de celle de l'équation 4.2 :

$$C_{max}^{opt} \geq max\{2np, (n - 2m)(L + 2p)\} \tag{4.3}$$

□

Borne supérieure obtenue par notre heuristique

Lemme 4.3.2 *L'heuristique basée sur la recherche d'un recouvrement des sommets de G_c par $\mathcal{K}$ cliques maximales donne une borne supérieure égale à $\mathcal{K}(L + p) + np$.*

Preuve :
La première idée qui découle naturellement au vu de la forme de la solution optimale dans un cas parfait, consiste à rechercher des cliques maximales dans G_c afin de combler le maximum de slots créés par les tâches-couplées. Chaque clique maximale va entraîner comme précédemment l'exécution d'un bloc de tâches-couplées, mais cette fois-ci le bloc ne sera pas sans temps d'inactivité. Pour calculer la longueur de l'ordonnancement obtenu nous sommons le nombre de blocs obtenus, créant chacun un slot de longueur L à remplir, et rajoutons le nombre de tâches à exécuter qui représente le temps séquentiel de toutes les sous-tâches b_i à exécuter (Voir Figure 4.9).

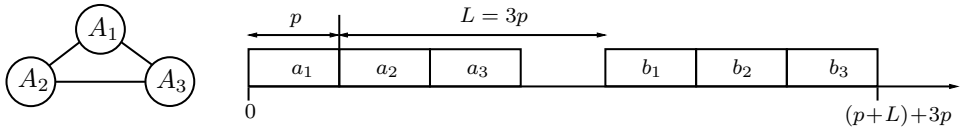


Fig. 4.9 – Exemple du calcul de la longueur d'un bloc

Le makespan obtenu pour un recouvrement des sommets de G_c par $\mathcal{K}$ cliques maximales donne la borne supérieure suivante :

$$C_{max}^{h} \leq \mathcal{K}(L + p) + \sum_{i=1}^{n} b_i = \mathcal{K}(L + p) + np \tag{4.4}$$

□

Étude de la performance relative ρ

Nous pouvons désormais calculer la performance relative du problème Π_1 que l'on obtient en utilisant cette heuristique.

Théorème 4.3.1 *Cette heuristique donne une performance relative $\rho \leq \frac{7p+L}{4p}$.*

Preuve :
En appliquant la Définition 2.1.23 de la page 13 sur le calcul de la performance relative, et en utilisant les bornes obtenues (données par les équations (4.3) et (4.4)), nous obtenons la performance relative suivante :

$$\rho \leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{\mathcal{K}(L+p)+np}{2np} \tag{4.5}$$

De ce ratio, nous pouvons analyser la valeur du rapport de la performance relative lorsque l'heuristique utilisée pour approximer le problème consiste à trouver un couplage maximum M de taille m. Dans ce cas là, $\mathcal{K} = (n-m)$ vu que le couplage crée m blocs de taille $(L+3p)$ et que les tâches isolées forment $(n-2m)$ blocs de taille $(L+2p)$. En remplaçant $\mathcal{K}$ dans la borne obtenue à l'équation (4.4), nous obtenons cette nouvelle borne supérieure :

$$C^h_{max} \leq (n-m)(L+p)+np \tag{4.6}$$

Grâce à l'étude du max de la borne inférieure (donnée par l'équation (4.2)), nous pouvons analyser le comportement de la performance relative. Puisque $C^{opt}_{max} \geq max\{2np, (n-2m)(L+2p)\}$, les cas suivants doivent être considérés :
- Pour $m \in [0, \frac{Ln}{2(2p+L)}[$, $C^{opt}_{max} \geq (n-2m)(L+2p)$
- Pour $m \in [\frac{Ln}{2(2p+L)}, \frac{n}{2}]$, $C^{opt}_{max} \geq 2np$

Selon les valeurs de m, nous obtenons une nouvelle borne supérieure pour notre heuristique, et une nouvelle borne inférieure pour un ordonnancement optimal (voir illustration Figure 4.10). Le ratio optimum s'obtient lorsque $m = \frac{Ln}{2(2p+L)}$, le calcul suivant nous donne la valeur recherchée :

$$\begin{aligned}
\rho &\leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{(n-\frac{Ln}{4p+2L})(L+p)+np}{2np} \\
\rho &= \frac{\frac{4p+2L}{4p+2L}(L+p) - \frac{L}{4p+2L}(L+p) + p\frac{4p+2L}{4p+2L}}{2p} \\
\rho &= \frac{(4p+2L)(L+p) - L(L+p) + 4p^2 + 2Lp}{2p(4p+2L)} \\
\rho &= \frac{8p^2+7pL+L^2}{8p^2+4pL} \leq \frac{7pL+L^2}{4pL} = \frac{7p+L}{4p} = \frac{7}{4} + \frac{L}{4p}
\end{aligned} \tag{4.7}$$

Notons que pour $m = 0$, $\rho = 1$ (c'est évident, puisque le graphe de compatibilité est un ensemble de tâches indépendantes), de plus pour $m = \dfrac{n}{2}$, $\rho = \frac{3p+L}{4p}$.

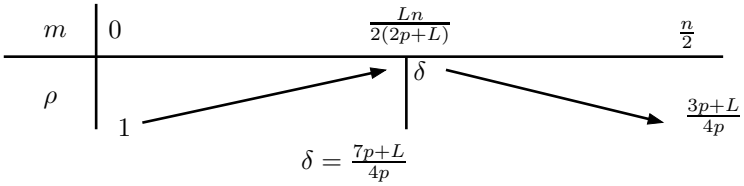


Fig. 4.10 – Comportement de la performance relative ρ en fonction de m

□

Ceci finit l'analyse du problème Π_1. Du côté négatif, nous avons montré que le problème était $\mathcal{NP}$-complet. Du côté positif, nous avons donné un algorithme d'approximation avec une performance relative bornée par $\rho < \frac{7p+L}{4p}$, où L et p sont des paramètres du problème. Le fait que la valeur de la performance relative ρ associée à l'algorithme dépende des paramètres L et p, nous amène à la question suivante : «Est-ce que notre problème admet un algorithme d'approximation avec une garantie de performance égale à une valeur constante, ou est-il non-APX[9] ?».

4.3.2 Étude d'approximation pour le problème $\Pi_2 : 1|a_i = L_i = b_i, G_c|C_{max}$

Nous allons faire l'étude de l'approximation d'une instance particulière du problème Π_2, lorsque les tâches d'acquisition $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ sont toutes différentes. Donc pour toutes tâches A_i et A_j, $i \neq j$, nous avons $a_i = L_i = b_i \neq a_j = L_j = b_j$. Cette hypothèse nous permet d'ordonner les tâches selon la taille L_i de chaque tâche (le paramètre L_i servira de repère pour différentier les tâches dans la suite), ce qui donne $L_1 \geq L_2 \geq \ldots \geq L_n$. Nous allons donner une borne inférieure et une borne supérieure obtenue avec un algorithme basé sur un recouvrement par cliques maximales des sommets de G_c.

Borne inférieure pour toute solution optimale

Lemme 4.3.3 *Une borne inférieure triviale basée sur le temps séquentiel pour* Π_2 *est égale à* $C^{opt}_{max} \geq \sum_{i=1}^{n} 2L_i$

Preuve :
L'ordonnancement optimal est obtenu lorsque nous n'avons aucun temps d'inactivité, ce cas ne peut arriver vu la structure de nos tâches et le fait qu'elles soient toutes différentes, mais nous choisissons délibérément une borne triviale. La borne inférieure de notre étude sera donc :

$$C^{opt}_{max} \geq T_{seq} = \sum_{i=1}^{n} 2L_i \tag{4.8}$$

□

Borne supérieure obtenue par notre heuristique

L'heuristique que nous proposons consiste à emboîter les tâches d'acquisition les unes dans les autres afin de minimiser le temps d'inactivité sur le processeur. Cette technique que nous appellerons par la suite «technique des poupées russes» nécessite de nouvelles définitions puis une transformation du graphe de compatibilité G_c.

Définition 4.3.1 (Tâches emboîtées) :
Deux tâches d'acquisition s'emboîtent l'une dans l'autre si et seulement si $a_i + L_i + b_i \leq L_j$ *ou* $a_j + L_j + b_j \leq L_i$.

Définition 4.3.2 (Graphe de compatibilité modifié) :
Soit G_c *un graphe de compatibilité quelconque. Nous notons* G^m_c *le graphe de compatibilité modifié où nous laissons uniquement les arêtes entre deux sommets* A_i *et* A_j *qui peuvent s'emboîter l'une dans l'autre.*

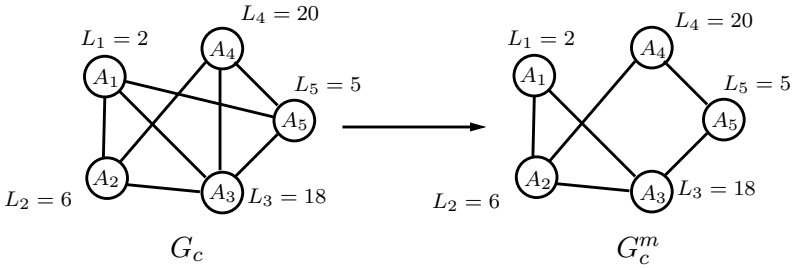


Fig. 4.11 – Exemple de modification du graphe de compatibilité G_c. Une arête $e = (A_i, A_j)$ est gardée lorsque $L_i \geq 3L_j$ ou $L_j \geq 3L_i$.

À partir du graphe de compatibilité modifié G_c^m obtenue, nous allons calculer la borne supérieure obtenue avec l'algorithme 5.

Algorithme 5 : Un algorithme d'approximation en temps polynomial pour Π_2 basé sur la recherche de clique maximale dans G_c^m

Données : $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, G_c
Résultat : C_{max}^h
début
- Le graphe G_c est modifié en G_c^m à la manière de la Définition 4.3.2
- **tant que** *il reste des sommets non visités* **faire**
  Pour un sommet i non visité de G_c^m, chercher une clique maximale. Tous les sommets de la clique maximale sont maintenant visités.
- À partir des cliques maximales obtenues, nous ordonnançons les tâches de chaque clique les unes dans les autres formant ainsi un bloc
- La somme des longueurs des blocs créés donne le makespan voulu C_{max}^h

fin

Lemme 4.3.4 *L'algorithme 5 basé sur la recherche d'un recouvrement des sommets du graphe modifié G_c^m par K cliques maximales donne une borne supérieure égale à $3\sum_{i=1}^{K} L_i$ pour Π_2.*

Preuve :
Chaque clique obtenue permet d'ordonnancer les tâches d'acquisition les unes dans les autres, le nombre de cliques obtenues indique le nombre de blocs créés par l'ordonnancement. La durée d'exécution de chaque bloc est alors égale à trois fois la durée L_i de la tâche qui contient toutes les autres tâches. Afin de calculer la borne supérieure associée à l'algorithme, nous cherchons le pire cas qui est lorsque pour K cliques maximales trouvées, les K tâches les plus grandes selon le paramètre L_i, sont réparties dans chaque clique du recouvrement. La taille de chaque bloc étant alors égal à trois fois un de ces L_i, la borne obtenue est forcément inférieur ou égale à :

$$C_{max}^h \leq 3\sum_{i=1}^{K} L_i \tag{4.9}$$

□

[9] La définition de non-APX est donnée à la Section 2.1.5 page 13.

Étude du rapport de la performance relative ρ

Nous allons calculer un premier rapport de performance relative du problème Π_2 que nous obtenons en utilisant cet algorithme.

Théorème 4.3.2 *Cet algorithme donne une performance relative $\rho \leq \frac{3}{2}$.*

Preuve :
En appliquant la définition 2.1.23 sur le calcul de la performance relative, et en utilisant les bornes obtenues (4.8) et (4.9), nous obtenons la performance relative suivante :

$$\rho \leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{3\sum_{i=1}^{K} L_i}{2\sum_{i=1}^{n} L_i} \leq \frac{3}{2} \tag{4.10}$$

□

Cette première borne d'approximation obtenue est assez triviale, dans la suite nous allons étudier des cas où cette borne sera plus faible selon la topologie du graphe de compatibilité.

Étude dans des topologies particulières

La difficulté de trouver un meilleur ratio nous amène à étudier les heuristiques d'approximation pour des graphes à structures particulières. Remarquons, par leur structure, que les tâches ne peuvent pas se chevaucher comme précédemment, mais peuvent s'emboîter à la manière de la technique « poupées russes ». Donc les seules arêtes à considérer sont celles qui relient une tâche A_i à une tâche A_j telles que $x_i \geq 3x_j$, où $a_i = b_i = L_i = x_i$ et $a_j = b_j = L_j = x_j$.

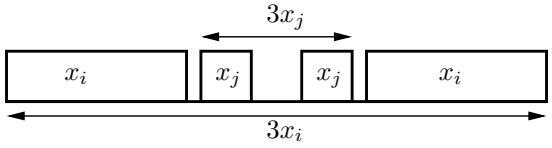


Fig. 4.12 – Illustration de l'emboîtement de deux tâches.

Donc à partir du graphe de compatibilité G_c, nous pouvons construire un graphe orienté ne contenant que les arêtes utiles et orientés en arc de A_i vers A_j, si A_j peut être exécuté entièrement durant l'exécution de A_i. Nous nous intéressons au cas où ce graphe est biparti complet orienté avec $V = X \cup Y$ et tel que les arcs sont orientés de X vers Y, donc toutes les tâches sont différentes.

Si nous notons X_i (resp. Y_i) les tâches de X (resp. de Y) pour $1 \leq i \leq k$ (resp. $1 \leq i \leq m$), et x_i (resp. y_i) leur durée, nous avons sans perte de généralité :

$$x_1 > x_2 > x_3 > \ldots > x_k \geq 3y_1 > 3y_2 > \ldots > 3y_m$$

Théorème 4.3.3 *Le problème de décider si nous pouvons trouver un ordonnancement en temps C_{max} pour le problème $1|a_i = L_i = b_i = x_i, G_c = biparti\ complet|C_{max}$ est $\mathcal{NP}-complet$.*

Preuve :
Nous ferons la preuve simplement dans le cas où le graphe biparti, noté $G^b_c = (X, Y, E)$, est tel que $|X| = 2$. Elle peut facilement être étendue pour $|X| > 3$. La preuve se fait à partir du

problème $\mathcal{NP}-$complet PARTITION PAIR[10] (problème 2.2.8). Nous contruisons une instance $\mathcal{I}^*$ à partir d'une instance $\mathcal{I}$ du problème PARTITION PAIR de la manière suivante :

Soit B la moitié de la somme des éléments de l'instance $\mathcal{I}$. Nous considérons deux tâches d'acquisition $X_1, X_2 \in X$, de longueur x_1 et x_2, et n autres tâches $Y_1, \ldots, Y_n \in Y$, de longueur $y_1, \ldots, y_n$, tels que :

- $X_1 = (3B+1, 3B+1, 3B+1)$ et $Y_2 = (3B, 3B, 3B)$
- $Y_i = (a_i = s(e_i), L_i = s(e_i), b_i = s(e_i)), \forall i \in \{1, \ldots, n\}$

Nous créons un graphe complet entre les sommets de X et de Y.

$\Rightarrow$ Supposons que nous avons une réponse oui au problème de PARTITION PAIR, montrons qu'il existe un ordonnancement de longueur $C_{max} = 18B + 3$. Nous exécutons les deux tâches X_1 et X_2 de manière contiguë. Il suffit ensuite d'exécuter dans le slot de taille $L = 3B$ ou $L = 3B + 1$ (associé aux tâches de X) les tâches d'acquisition associées aux objets de la partition de manière contiguë.

$\Leftarrow$ Réciproquement, nous supposons que nous avons un ordonnancement de longueur $C_{max} = 18B + 3$. Montrons l'existence d'une partition :

Les deux tâches d'acquisition associées aux sommets de X ne peuvent pas s'exécuter l'une dans l'autre. Il est clair que la somme des durées d'exécution de ces deux tâches est de $18B + 3$. Donc toutes les autres tâches d'acquisition doivent s'exécuter dans les slots de longueurs $3B$ ou $3B + 1$. Ce qui entraîne l'existence d'une solution pour PARTITION PAIR.

□

Maintenant que nous avons la $\mathcal{NP}-$complétude du problème, nous pouvons l'étudier du point de vue de l'approximation. Montrons que l'algorithme glouton 6 est dans ce cas $\frac{7}{6}$-approché.

Algorithme 6 : Algorithme glouton similaire à un Best-Fit

Données : $G_c^b = (X \cup Y, E)$, X, Y
Résultat : C_{max}^h longueur de l'ordonnancement obtenue avec notre heuristique
début
  Trier les tâches de X par ordre décroissant des x_i, $\forall i = 0, \ldots, |X|$
  Trier les tâches de Y par ordre décroissant des y_i, $\forall i = 0, \ldots, |Y|$
  Exécuter les tâches de X consécutivement selon l'ordre décroissant précédent
  tant que *il reste des tâches de* Y *non exécutées* **faire**
    si *la première tâche restante, selon l'ordre décroissant précédent, peut s'exécuter dans le temps d'inactivité de la première tâche* X **alors**
      l'exécuter dans le temps d'inactivité de cette tâche de X
    sinon
      essayer avec les tâches suivantes de X
fin

Cet algorithme est similaire à la stratégie Best-Fit, la complexité de notre algorithme dépend essentiellement du tri des deux ensembles de tâches X et Y. La complexité sera donc au pire d'ordre $O(nlog(n))$, où $|X| + |Y| = n$.

Avant de donner le Théorème 4.3.4, nous définissons le terme suivant :

[10] Équivalent au problème PARTITION mais dans lequel les éléments e_i sont tous pairs.

Définition 4.3.3 *On appelle résidus, les tâches d'acquisition de type Y qui ne sont pas exécutées dans le slot des tâches d'acquisition de type X.*

Théorème 4.3.4 *L'algorithme 6 a une performance relative de $7/6$ pour le problème $1|a_i = L_i = b_i = x_i, G_c = \text{biparti complet}|C_{max}$.*

Preuve :
Nous supposons qu'il y a un résidu R sinon l'algorithme donne forcément la solution optimale. Nous notons $C_1 = \sum_{i=1}^{m} 3y_i$ et $C_2 = \sum_{i=1}^{K} x_i$.

Nous allons utiliser le Lemme 4.3.5 pour la fin de la preuve :

Lemme 4.3.5 *Le slot de taille x_i, $\forall X_i \in X$, est rempli au moins à la moitié par l'exécution de tâches de Y s'il y a un résidu.*

Preuve :
Raisonnons par l'absurde. Soit X_i la première tâche non remplie à la moitié, et Y_j la première tâche du résidu. On a $y_j > \frac{x_i}{6}$ sinon il y aurait la place d'exécuter Y_j dans la place disponible du slot de X_i (par hypothèse au moins à moitié vide). D'après notre algorithme, nous prenons les tâches de Y par ordre décroissant, du coup nous avons $y_1 > \ldots > y_j > \frac{x_i}{6}$. Ainsi, le slot de X_i est forcément vide lorsque nous plaçons Y_j à la suite des tâches de X, et donc $y_j > \frac{L_i}{3}$ (ce qui est en contradiction avec le fait que G_c est biparti complet orienté, et donc que toute tâche de Y est exécutable dans celles de X).

□

Suite de la preuve du Théorème 4.3.4 :

On a donc $0 \leq R \leq C_1 - \frac{C_2}{2}$ et nous obtenons la borne supérieure :

$$\begin{aligned} C^h_{max} &\leq \text{Résidu} + \text{la durée des tâches de } X \text{ exécutées séquentiellement} \\ &= \left(C_1 - \frac{C_2}{2}\right) + 3C_2 = \frac{5C_2}{2} + C_1 \end{aligned} \tag{4.11}$$

Nous avons alors deux cas possibles :
- Si $C_1 \leq C_2$ alors $C^{opt}_{max} \geq 3C_2$, en effet au mieux les tâches de Y sont exécutées dans les slots des tâches de X.
- Si $C_1 > C_2$ alors alors $C^{opt}_{max} \geq 2C_2 + C_1$, en effet au mieux les slots des tâches de X sont utilisées complètement.

La performance relative vaut alors :

$$\begin{aligned} \rho = \frac{C^h_{max}}{C^{opt}_{max}} &\leq \frac{\frac{5C_2}{2} + C_1}{2C_2 + C_1} \leq \frac{\frac{5C_2}{2} + y}{2C_2 + y} \quad \forall y \text{ minorant de } A \\ \frac{C^h_{max}}{C^{opt}_{max}} &\leq \frac{7}{6} \quad \text{en prenant } C_2 \text{ comme minorant de } C_1 \end{aligned}$$

□

Remarque 4.3.1 *Notre analyse pour obtenir une performance relative de $\frac{7}{6}$ n'est pas assez «fine», dans notre étude du pire cas nous avons fait des majorations rapides. Il semble difficile d'exhiber une instance pour laquelle cette borne est atteinte. Nous avons construit une instance, donnée sur l'Exemple 4.3.1, induisant une borne atteinte de $\frac{8}{7}$. Nous pensons que cette borne est la vraie borne d'approximation pour notre algorithme.*

Exemple 4.3.1 *Soient quatre tâches d'acquisition $A_1 \in X$, $A_2 \in Y$, $A_3 \in Y$ et $A_4 \in Y$ définies de la manière suivante :*

- $X_1 = (M, M, M)$ *où* $M \in \mathbb{N}^*$
- $X_2 = (\frac{M}{6} + 2\epsilon, \frac{M}{6} + 2\epsilon, \frac{M}{6} + 2\epsilon)$ *où* $M \in \mathbb{N}^*$
- $X_3 = (\frac{M}{6} + \epsilon, \frac{M}{6} + \epsilon, \frac{M}{6} + \epsilon)$ *où* $M \in \mathbb{N}^*$
- $X_4 = (\frac{M}{6} - \epsilon, \frac{M}{6} - \epsilon, \frac{M}{6} - \epsilon)$ *où* $M \in \mathbb{N}^*$

En appliquant l'Algorithme 6, nous obtenons un ordonnancement de longueur $\boldsymbol{C^h_{max} = 4M}$ (voir Figure 4.13). La solution optimale consiste à exécuter côte à côte A_3 et A_4 dans le temps d'inactivité de la tâche A_1 (voir Figure 4.14), nous obtenons $\boldsymbol{C^{opt}_{max} = \frac{7M}{2}}$. La performance relative obtenue est donc $\boldsymbol{\rho \leq \frac{8}{7}}$.

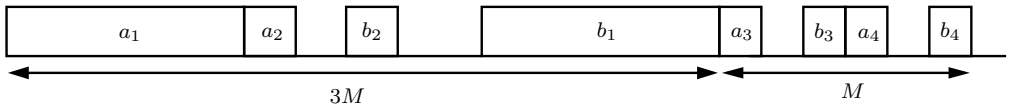


Fig. 4.13 – Ordonnancement obtenu avec l'Algorithme 6

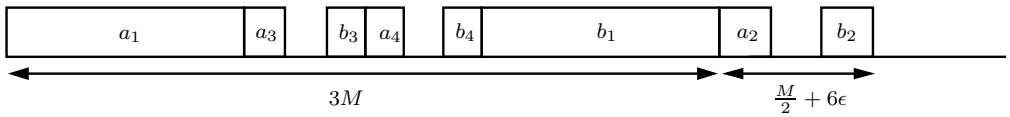


Fig. 4.14 – Ordonnancement optimal

4.3.3 Étude d'approximation pour le problème Π_3

L'étude de la complexité de ce problème a été faite à la Section 4.2.2 page 45. Nous avons vu que le problème était polynomial lorsque $L < a + b$, et $\mathcal{NP}$-complet lorsque $L \geq a + b$. Nous nous sommes intéressés au cas le plus critique lorsque $L = a + b$, cette démarche est toujours motivée par la volonté de déterminer les paramètres responsables du passage de la polynomialité à la $\mathcal{NP}$-complétude.

Approximation dans le cas particulier où $L_i = a + b$

Dans la sous-Section 4.2, nous avons étudié la complexité du problème Π_3 lorsque $L_i = a+b$. Nous avons montré la relation entre minimiser le makespan et partitionner le graphe G_c en chaînes disjointes $\{C_1, C_2, \ldots, C_n\}$ (les chaînes de longueur 0 et 1 sont acceptées). Le makespan est égal au temps séquentiel plus le temps d'inactivité créé par chaque chaîne (voir figure 4.6), qui est de $(a + b)$ pour tout chaîne de longueur strictement supérieure à 1 ou égale à 0, et de a pour les chaînes de longueur 1 (les arêtes).

Ainsi, trouver un ordonnancement optimal peut être considéré comme un problème de graphe que nous appelons **COUVERTURE MINIMUM EN CHAÎNES DISJOINTES**

LIÉE À UN ORDONNANCEMENT (Min-CCDLO) et défini de la manière suivante :

Données	:	Un graphe $G = (V, E)$ d'ordre n, deux entiers naturels a et b tels que $b < a$.
Résultat	:	Une partition $\mathcal{P}$ des sommets de G en chaînes disjointes (qui peuvent être de taille 0).
Objectif	:	Minimiser $n(a+b) + \sum_{p \in \mathcal{P}} w(p)$ où $w : \mathcal{P} \to \mathbb{N}$ est une fonction de poids telle que $w(p) = a$ si et seulement si $\|E(p)\| = 1$, et $w(p) = a + b$ dans le cas contraire.

Problème d'optimisation 4.3.1: COUVERTURE MINIMUM EN CHAÎNES DISJOINTES LIÉE À UN ORDONNANCEMENT (Min-CCDLO)

Dans n'importe quelle solution, chaque chaîne augmente le coût du temps d'inactivité par au moins a (quand la chaîne a une longueur 1), et au plus $a + b < 2a$. Nous pouvons ainsi en déduire qu'une solution optimale de Min-CCDLO, lorsque $b < a$, consiste à trouver une partition $\mathcal{P}$ d'une certaine cardinalité k^*, et un nombre maximum de chaînes de longueur 1 parmi toutes les k^*-partitions possibles.

Pour commencer, nous allons montrer que n'importe quel algorithme[11] d'approximation en temps polynomial admet une garantie de performance d'au plus 2 pour le problème Min-CDDLO. Puis nous développerons un algorithme $\frac{3}{2}$-approximable en temps polynomial basé sur un couplage maximum dans le graphe G_c. En fait, nous montrons que cet algorithme a un ratio d'approximation d'au plus $\frac{3a+2b}{2a+2b}$, qui amène à avoir un ratio entre $\frac{3}{2}$ et $\frac{5}{4}$ selon les valeurs de a et b (avec $b < a$). Ce résultat, qui dépend des valeurs de a et b, sera développé et discuté plus loin dans le but de proposer un meilleur ratio selon le type de graphe étudié.

À toute instance de Min-CCDLO, nous pouvons associer un ordonnancement ayant pour makespan $C^{opt}_{max} = T_{seq} + T^{opt}_{idle}$ où $T_{seq} = n(a + b)$.

Remarque 4.3.2 *Pour toute solution obtenue par une heuristique h de coût C^h_{max}, dont l'objectif est de minimiser les temps d'inactivité, nous avons nécessairement*[12] $T^h_{idle} \geq (a + b)$ *et également*[13] $T^h_{idle} \leq n(a + b)$. *Ainsi, pour n'importe quelle solution h de Min-CCDLO, nous avons le ratio de la performance relative ρ égale à :*

$$\rho \leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{2n(a+b)}{(n+1)(a+b)} < 2. \tag{4.12}$$

Dans la suite, nous allons développer un algorithme d'approximation en temps polynomial basé sur un couplage maximum dans le graphe G_c, avec une garantie de performance située dans l'intervalle $[\frac{5}{4}, \frac{3}{2}]$ selon les valeurs de a et b.

[11] Entre deux tâches indépendantes, il n'est pas permis d'avoir un temps d'inactivité lorsqu'il y a des tâches exécutables.

[12] L'égalité est obtenue lorsque le graphe G_c possède un chemin hamiltonien, autrement nous avons besoin d'au moins deux chaînes pour couvrir G_c (lorsque G_c n'est pas seulement une arête), qui amène à augmenter T^h_{idle} par au moins $2a \geq a + b$ unités de temps.

[13] Le pire cas consiste à exécuter séquentiellement les tâches sans chercher à ordonnancer une sous-tâche a_j ou b_j d'une tâche A_j durant le slot d'inactivité d'une tâche A_i.

Soit I une instance de notre problème. Une solution optimale a la forme d'un recouvrement par des chaînes disjointes. Les n sommets sont partitionnés en trois ensembles disjoints : n_1 sommets non-couverts, n_2 sommets couverts par $\alpha_2 = \frac{n_2}{2}$ chaînes de longueur 1, et enfin n_3 sommets couverts par exactement α_3 chaînes de longueur strictement plus grande que 1 (voir l'illustration Figure 4.15). La borne inférieure associée à la solution optimale est égale à : $C^{opt}_{max} = n(a+b)+n_1(a+b)+\frac{n_2}{2}a+\alpha_3(a+b)$.

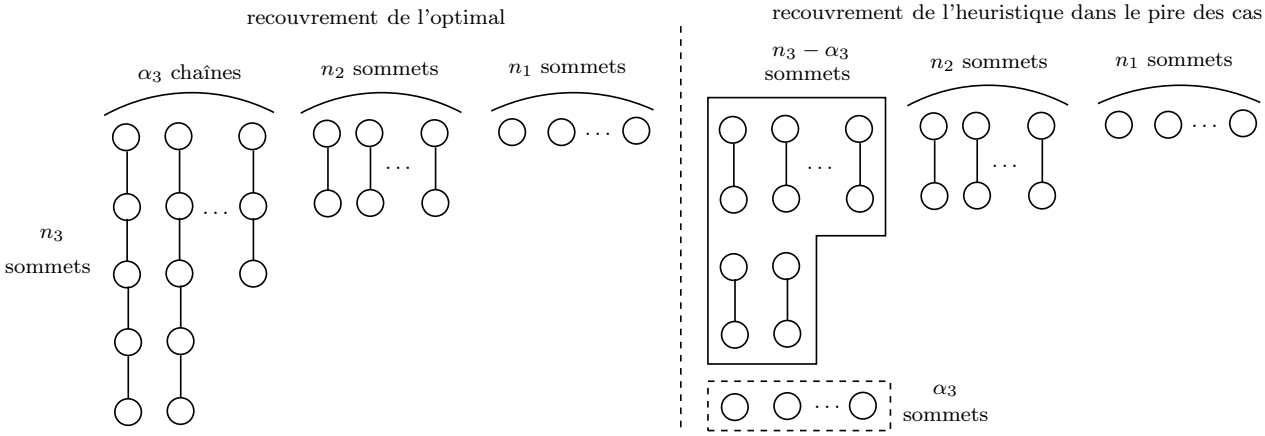


Fig. 4.15 – Illustration de la solution optimale pour une instance I

Maintenant nous allons proposer un algorithme d'approximation en temps polynomial avec un ratio non trivial pour une instance I. Cet algorithme est basé sur un couplage maximum G_c dans le but d'exécuter deux sous-tâches à la fois. Pour deux tâches-couplées A_i et A_j reliées par une arête du couplage, nous obtenons un temps d'inactivité de longueur a (voir Figure 4.6(b) page 47).

Avec cet algorithme et à partir du découpage fait pour une solution optimale (voir Figure 4.15), nous allons calculer le nombre d'arêtes obtenu par le couplage maximum selon le recouvrement de la solution optimale. Nous savons qu'il y a au moins n_1 sommets non recouverts (resp. n_2 sommets couverts par des arêtes) par le couplage maximum. Pour les α_3 chaînes, nous considérons le pire cas dans lequel toutes les chaînes sont de longueur impaire. Ainsi, à partir des n_3 sommets composant ces chaînes, le couplage maximum de l'algorithme va recouvrir dans le pire des cas exactement $(n_3 - \alpha_3)$ sommets et laissera α_3 sommets non recouverts. La borne supérieure est alors égale à :

$$\begin{aligned} C^h_{max} &\leq n(a+b) + n_1(a+b) + \frac{n_2}{2}a + \alpha_3(a+b) + (n_3-\alpha_3)\frac{a}{2} \\ &\leq \frac{3na}{2} + nb + \frac{\alpha_3 a}{2} + \alpha_3 b + \frac{n_1 a}{2} + n_1 b \end{aligned}$$

Il est facile de voir que le pire cas apparaît lorsqu'il existe une chaîne hamiltonienne, ce qui donne $n_1 = n_2 = 0$, $\alpha_3 = 1$ et $n_3 = n$. La performance relative obtenue est alors égale à :

$$\rho \leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{n(a+b) + (a+b) + (n-1)\frac{a}{2}}{n(a+b) + (a+b)} \leq \frac{3a+2b}{2a+2b} \tag{4.13}$$

Étude d'instances avec des topologies particulières

Nous concluons l'étude d'approximation sur le problème Min-CCDLO lorsque G_c admet une topologie particulière. Clairement, min-CCDLO est équivalent à Π_3 avec $b < a$ et $L = a + b$, et peut être vu comme la formulation d'un problème d'ordonnancement en problème de graphe. Ce problème est très proche du problème (2.2.6) bien connu **COUVERTURE MINIMUM PAR DES CHAÎNES DISJOINTES (Min-CCD)** qui consiste à recouvrir les sommets d'un graphe avec un minimum de chaînes disjointes[14]. Ce problème a été largement étudié et nous avons donné un état de l'art sur les principaux travaux réalisés dans la littérature dans la Section 3.3.4.

Le problème Min-CDD est directement lié au problème de COMPLÉTION HAMILTONIEN noté (COMPH) [31], qui consiste à trouver le nombre minimum d'arêtes, noté $COMPH(G)$, qui doit être ajouté à un graphe G donné, dans le but de le rendre hamiltonien (de garantir l'existence d'un cycle hamiltonien). Et il a été montré que si G n'est pas hamiltonien, alors la cardinalité d'un recouvrement minimum par des chaînes disjointes est clairement égale à $COMPH(G)$.

Le dual de Min-CDD est connu dans la littérature [31] sous le nom de RECOUVREMENT MAXIMUM PAR DES CHAÎNES DISJOINTES. Il consiste à trouver dans G une collection de chaînes disjointes de longueur au moins un, qui maximise les arêtes couvertes dans G. Ce problème est connu pour être $\frac{7}{6}$-approximable [10].

Nous allons montrer que la recherche d'une ρ_{CCD}-approximation pour Min-CCD sur G_c permet de trouver pour Min-CDDLO une stratégie avec un ratio de performance égale à $\rho_{CDDLO} \leq \min\{\rho_{CCD} \times (\frac{a+b}{a}), \frac{3a+2b}{2a+2b}\}$. Ce qui nous amènera à avoir, indépendamment des valeurs de a et b, une $\frac{1+\sqrt{3}}{2}$-approximation pour Min-CCDLO lorsque Min-CCD peut être résolu en temps polynomial sur G_c.

Dans la littérature, nous avons vu que Min-CCD était polynomial pour les arbres [36, 44, 20], les graphes de permutation biparti [65], les cactis [54], et bien d'autres classes encore. Il n'y a actuellement aucun résultat sur la complexité de Min-CCDLO sur de tels graphes, puisque du fait que les valeurs de a et b ont une forte influence sur les calculs, les techniques utilisées pour prouver la polynomialité de Min-CCD ne peuvent être adaptées à la preuve de la polynomialité de Min-CCDLO. En dépit de nos efforts, la complexité de Min-CCDLO reste un problème ouvert. Toutefois, en utilisant les résultats sur Min-CCD, nous montrons comment le ratio d'approximation peut être diminué pour ces classes de graphes où Min-CCD est polynomial.

Lemme 4.3.6 *Si Min-CCD peut être résolu en temps polynomial, alors il existe un algorithme polynomial donnant une* $\frac{(a+b)}{a}$*-approximation pour Min-CCDLO.*

Preuve :
Soit $I_1 = (G)$ une instance de Min-CCD, et $I_2 = (G, a, b)$ une instance de Min-CCDLO. Soit $\mathcal{P}_1^*$ la partition en chaînes disjointes correspondante à une solution optimale de Min-CCD de coût $|\mathcal{P}_1^*|$, et $\mathcal{P}_2^*$ celle correspondante à une solution optimale de Min-CCDLO de coût OPT_{CCDLO}. D'après la définition de Min-CCDLO (4.3.1), nous avons $|\mathcal{P}_2^*| \geq |\mathcal{P}_1^*|$[15]. Puisque chaque chaîne, utilisée par une solution pour Min-CCDLO, augmente le coût de la solution d'au moins a, nous

[14] Parfois référencé comme PARTITION EN CHAÎNES.

[15] La meilleure solution pour Min-CCDLO n'est pas nécessairement une solution minimisant le nombre de chaînes disjointes.

avons alors :

$$OPT_{CCDLO} = \sum_{p\in\mathcal{P}_2^*} w(p) + n(a+b) \geq a|\mathcal{P}_2^*| + n(a+b) \geq a|\mathcal{P}_2^*| \quad (4.14)$$

$$\Rightarrow \frac{OPT_{CCDLO}}{a} \geq |\mathcal{P}_2^*| \quad (4.15)$$

Analysons le coût donné par la partition $\mathcal{P}_1^*$ si nous la considérons comme une solution (pas nécessairement optimale) pour l'instance I_2 de Min-CCDLO. Puisque chaque chaîne, utilisée par une solution pour Min-CCDLO, augmente le coût de la solution d'au plus $(a+b)$, nous avons alors :

$$\begin{aligned}
\sum_{p\in\mathcal{P}_1^*} w(p) + n(a+b) &\leq (a+b)|\mathcal{P}_1^*| + n(a+b) \leq (a+b)|\mathcal{P}_2^*| + n(a+b) \\
&\leq b|\mathcal{P}_2^*| + OPT_{CCDLO} \qquad \text{d'après (4.14)} \\
&\leq \frac{b}{a}OPT_{CCDLO} + OPT_{CCDLO} \qquad \text{d'après (4.15)} \\
&\leq \frac{a+b}{a}OPT_{CCDLO} \quad (4.16)
\end{aligned}$$

□

La même preuve peut être appliquée s'il existe une ρ_{CCD}-approximation pour Min-CCD, car il existe alors une $\rho_{CCD} \times (\frac{a+b}{a})$-approximation pour Min-CCDLO. Supposons que nous connaissions une constante ρ_{CCD} telle qu'il existe une ρ_{CCD}-approximation pour Min-CCD.

Soit S_1 la stratégie qui consiste à déterminer une $\rho_{CCD} \times (\frac{a+b}{a})$-approximation pour Min-CCDLO à partir de ρ_{CCD}, et S_2 la stratégie, qui consiste à utiliser l'algorithme présenté précédemment basé sur un couplage. Clairement, S_1 est particulièrement efficace lorsque b est très petit en comparaison de a, tandis que S_2 donne un meilleur ratio lorsque b est proche de a. Ces deux stratégies sont complémentaires selon les valeurs de a et b, qui varient de 0 à a. Le fait de choisir le meilleur résultat entre l'exécution de S_1 et S_2, donne un ratio de performance ρ_{CCDLO} tel que :

$$\rho_{CCDLO} \leq \min\left\{\rho_{CCD} \times \left(\frac{a+b}{a}\right), \frac{3a+2b}{2a+2b}\right\}. \quad (4.17)$$

Comparé à l'exécution seule de S_1, cette stratégie améliore les résultats obtenus si et seulement si ρ_{CCD} est plus petit que $\frac{3}{2}$ (voir Figure 4.16).

Nous proposons alors la remarque suivante qui n'est pas une bonne nouvelle :

Remarque 4.3.3 *Il n'y a pas de ρ_{CCD}-approximation pour Min-CCD dans des graphes quelconques pour $\rho_{CCD} < 2$.*

Ce résultat est une conséquence du théorème de l'impossibilité [21]. Considérons une instance de Min-CCD possédant une chaîne hamiltonienne. La solution optimale de Min-CCD est une chaîne de longueur 1, alors pour n'importe quel algorithme ρ_{CCD}-approximable en temps polynomial avec $\rho_{CCD} < 2$, la solution obtenue sera inférieure à deux chaînes, donc égale à 1. Ce qui est impossible sous l'hypothèse que $\mathcal{P} \neq \mathcal{NP}$.

Ce résultat implique également que les algorithmes d'approximation à facteur constant pour Max-CCD ne donnent pas forcément la même garantie de performance pour Min-CCD, puisque le meilleur ratio d'approximation pour Max-CCD est $\frac{7}{6}$, qui est plus petit que la borne d'inapproximation de Min-CCD.

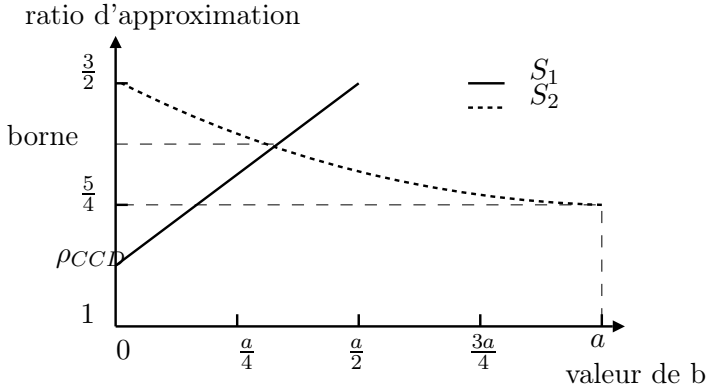


Fig. 4.16 – Trouver une bonne ρ_{CCD}-approximation aide à augmenter les résultats d'approximation de Min-CCDLO

La bonne nouvelle est que Min-CCD est polynomial sur nombre de classes de graphes [36, 44, 20, 65, 54], et donc $\rho_{CCD} = 1$. Pour tous ces graphes, nous obtenons un ratio d'approximation égal à $\min\{\frac{(a+b)}{a}, \frac{3a+2b}{2a+2b}\}$ qui est maximum lorsque $\frac{(a+b)}{a} = \frac{3a+2b}{2a+2b}$, c'est à dire :

$$\frac{(a+b)}{a} = \frac{3a+2b}{2a+2b} \Leftrightarrow -a^2 + 2ab + 2b^2 = 0. \tag{4.18}$$

La seule solution de cette équation avec a et $b \geq 0$ est $a = b(1+\sqrt{3})$. En remplaçant a par cette nouvelle valeur dans $\frac{(a+b)}{a}$ ou dans $\frac{3a+2b}{2a+2b}$, nous montrons que dans le pire cas le ratio d'approximation est réduit de $\frac{3}{2}$ à $\frac{1+\sqrt{3}}{2} \approx 1.37$.

4.4 Conclusion

Nous avons étudié tout au long de ce chapitre les problèmes d'ordonnancement sur mono processeur avec tâches d'acquisition en présence d'un graphe quelconque de compatibilité G_c. Les différents problèmes rencontrés viennent du fait que nous faisons varier les paramètres fondamentaux des tâches d'acquisition (a_i, L_i, b_i) de la même manière que le font Orman et Potts dans leur papier sur l'étude des tâches-couplées en présence d'un graphe de compatibilité complet.

Le but recherché tout au long du chapitre a été de déterminer l'impact de la contrainte d'incompatibilité sur ces problèmes, et analyser les cas critiques se trouvant à la limite entre la polynomialité et la $\mathcal{NP}$-complétude selon la valeur des paramètres (voir Figure 4.17).

Le chapitre est découpé en deux parties, la première portant sur l'étude de la complexité des problèmes critiques et par conséquent tous les problèmes plus généraux, et la deuxième partie portant sur l'étude de l'approximation de ces mêmes problèmes.

Nous donnons trois preuves de $\mathcal{NP}$-complétude pour les problèmes notés Π_1, Π_2 et Π_3, et deux preuves de polynomialité pour les problèmes Π_4 et Π_5 (Voir Figure 4.17). Les preuves sont basées sur le graphe de compatibilité G_c utilisant des réductions à partir de problèmes connus de recouvrement de sommets dans des graphes (PARTITION EN TRIANGLES, CHAÎNE HAMILTONIENNE). À partir de ces résultats nous en déduisons la $\mathcal{NP}$-complétude de tous les problèmes plus généraux.

Le premier constat que nous pouvons faire à la suite de ces résultats est que la contrainte d'incompatibilité entraîne la $\mathcal{NP}$-complétude de deux problèmes, l'un polynomial et l'autre

ouvert. Une étude plus approfondie du problème Π_3 permet de voir la limite entre la polynomialité et la $\mathcal{NP}$-complétude selon les valeurs des paramètres fondamentaux.

Pour ce qui est des preuves de polynomialité des problèmes notés Π_4 et Π_5, les algorithmes en temps polynomial donnés sont basés sur des **couplages maximum** ou **couplage parfait de poids maximum**. À partir de ces résultats nous en déduisons la polynomialité de tous les problèmes plus spécifiques.

La visualisation globale de la complexité des problèmes est représentée sur la Figure 4.17, nous avons mis en avant l'évolution de la complexité des problèmes lors de l'introduction de la contrainte d'incompatibilité.

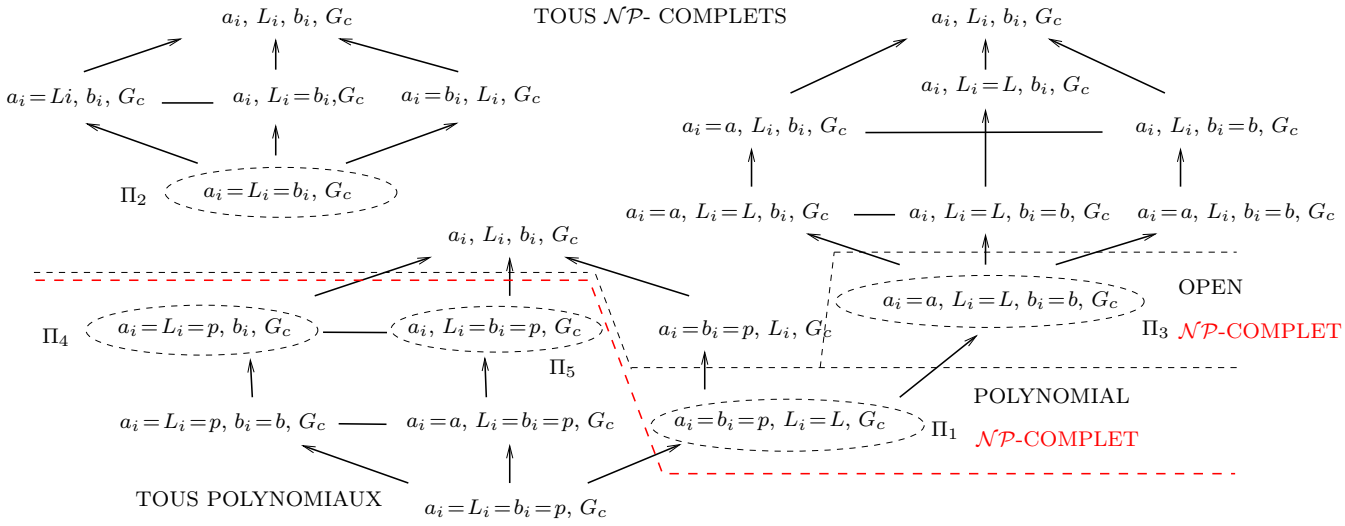


Fig. 4.17 – Visualisation globale de l'impact de l'introduction de la contrainte d'incompatibilité sur la complexité des problèmes d'ordonnancement avec tâches d'acquisition sur mono processeur. La ligne en pointillé noire représente les résultats sans la contrainte d'incompatibilité, et la ligne en pointillé rouge lorsque nous l'introduisons.

Il est intéressant d'observer que la complexité des problèmes dépend énormément du lien entre le paramètre L_i et l'un des deux autres : a_i ou b_i. Lorsque L_i est égal à a_i ou b_i, la seule façon d'ordonnancer les tâches est soit de les faire se chevaucher deux par deux, soit de les exécuter consécutivement. Cette configuration se répercute sur le graphe de compatibilité par la recherche de couplage maximum ou parfait. Par contre dès que L_i est indépendant des deux autres paramètres, les possibilités d'ordonnancement des tâches amènent à rechercher des recouvrements par chaînes, ou par cliques dans G_c, et la plupart de ces problèmes sont connus pour être $\mathcal{NP}$-complets.

La deuxième partie sur l'approximation permet de mieux apprécier le rôle du graphe de compatibilité G_c, nous proposons trois types d'heuristiques polynomiales pouvant s'appliquer à ces problèmes selon les paramètres fondamentaux. Ces heuristiques nous donnent des bornes d'approximation non triviales dépendant la plupart du temps du paramètre L_i.

La recherche d'heuristique efficace dépend énormément du graphe G_c, et les problèmes polynomiaux connus que nous avons utilisés sont :

- Le couplage de taille maximum
- Le recouvrement par cliques maximales

Dans une recherche plus approfondie du problème Π_3, nous avons étudié son approximation en fonction des résultats existants sur les problèmes de recouvrement par chaînes disjointes, et cela nous a permis d'obtenir une meilleure borne.

Le constat général que nous pouvons faire sur l'approximation des problèmes étudiés est le suivant : l'introduction de la contrainte d'incompatibilité **change fondamentalement** l'approche classique sur ce type de problème, et amène à étudier des problèmes de graphe, connus pour être difficilement approximables. Les bornes d'approximation obtenues dépendent pour la plupart du paramètre L_i, ce qui nous amène à nous poser cette question : « Les problèmes étudiés sont-ils approximables avec une garantie de performance égale à une constante ou sont-ils NON-APX ? ».

L'ensemble des résultats obtenus dans ce chapitre sont récapitulés et référencés dans le tableau 4.1.

Problème	Complexité	Ratio d'approx.	Référence
$\Pi_1 : (a_i = b_i = p, L_i = L), G_c$	$\mathcal{NP}$-complet	$\frac{7p+L}{4p}$	Page 52
$\Pi_2 : (a_i = L_i = b_i), G_c$	$\mathcal{NP}$-complet	$\frac{3}{2}$	Page 56
$\Pi_3 : (a_i = a, L_i = L = a+b, b_i = b), G_c$	$\mathcal{NP}$-complet	$\left[\frac{3}{2}, \frac{5}{4}\right]$	Page 59
$\Pi_4 : (a_i = L_i = p, b_i), G_c$	Polynomial	1	Page 47
$\Pi_5 : (a_i, L_i = b_i = p), G_c$	Polynomial	1	Page 50

TAB. 4.1 – Résumé des résultats du Chapitre 4

Chapitre 5

Prise en compte de tâches de traitement

5.1 Introduction

Ce chapitre portera sur l'étude des problèmes d'ordonnancement avec tâches d'acquisition en présence d'un **graphe de compatibilité complet** et d'un **graphe de précédence en ajoutant les tâches de traitement**. Le graphe de précédence aura la particularité suivante : chaque tâche d'acquisition sera le prédécesseur d'une tâche de traitement. De la même manière que dans la partie précédente, nous allons nous baser sur les résultats d'Orman et Potts [57] pour réaliser le même type d'étude en présence des tâches de traitement et d'un graphe de compatibilité complet.

En reprenant la notation de la Figure 3.5 page 33, nous présentons sur la Figure 5.1 la vision globale de la complexité des mêmes problèmes d'ordonnancement avec la présence de tâches de traitement. Les problèmes sont représentées à travers trois treillis comme précédemment où nous ajoutons la notation $\cup\boldsymbol{\mathcal{T}_i}$ pour indiquer la précence des tâches de traitement, et donc du graphe de précédence. Les problèmes étudiés dans ce chapitre seront notés Π_i^P.

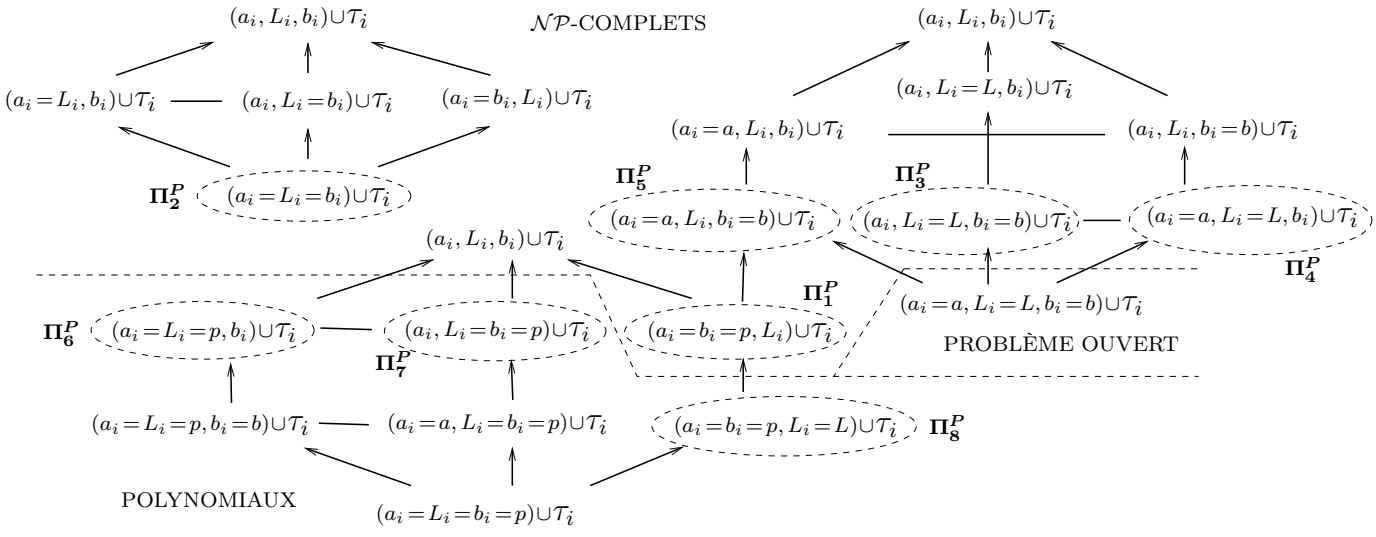


Fig. 5.1 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition en présence de contrainte de précédence

Les tâches de traitement ont la particularité d'être préemptives, ce qui permet d'utiliser les

temps d'inactivité créés par les tâches d'acquisition. Cette contrainte ne change pas la complexité des problèmes des treillis de la Figure 3.5, mais les algorithmes, permettant de résoudre en temps polynomial les problèmes Π_6^P, Π_7^P et Π_8^P, utilisent des techniques plus complexes.

En considérant la hiérarchie entre nos problèmes, il nous suffit d'étudier certains cas particuliers pour obtenir la complexité de tous les autres problèmes. Nous focaliserons donc notre étude sur les problèmes Π_i^P pour tout $i = 1, \ldots, 8$ (définis à la Figure 5.1), dont la complexité se situe à la frontière de la $\mathcal{NP}$-complétude et la polynomialité selon le schéma 5.1. Pour ces problèmes là, nous fixerons certains paramètres afin de mieux évaluer l'influence des tâches de traitement sur le passage de la polynomialité à la $\mathcal{NP}$-complétude.

5.1.1 Présentation du chapitre

Dans la première section nous donnerons les preuves de $\mathcal{NP}$-complétude et de polynomialité des huit problèmes les plus critiques, suivies d'une analyse plus poussée selon les valeurs des paramètres fondamentaux. Puis dans la deuxième section, nous donnerons plusieurs heuristiques pour les différents problèmes étudiés, en prenant en compte à nouveau des valeurs des paramètres selon les cas.

5.2 Classification de problèmes par la complexité

Ces problèmes généralisent ceux déjà étudiés par Orman et Potts (voir Figure 3.5 page 33) lorsque $\forall i\ \tau_i = 0$. Donc les résultats déjà vus $\mathcal{NP}$-complets restent $\mathcal{NP}$-complets avec l'introduction des tâches de traitement et donc du graphe de précédence. Néanmoins, nous allons montrer que la $\mathcal{NP}$-complétude est conservée même si $\forall i\ \tau_i \geq 1$, les cinq problèmes critiques qui nous intéressent sont :

- $\Pi_1^P : 1|prec, (a_i = b_i = p, L_i) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$ (Section 5.2.1 page 69)
- $\Pi_2^P : 1|prec, (a_i = b_i = L_i) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$ (Section 5.2.2 page 71)
- $\Pi_3^P : 1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$ (Section 5.2.3 page 72)
- $\Pi_4^P : 1|prec, (a_i = a, L_i = L, b_i) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$ (Section 5.2.4 page 72)
- $\Pi_5^P : 1|prec, (a_i = a, L_i, b_i = b) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$ (Section 5.2.5 page 72)

Les preuves données sont similaires à celles d'Orman et Potts [57] lorsqu'il n'y a pas de tâches de traitement. Afin d'alléger la lecture du chapitre, nous ne donnerons que la preuve pour Π_1, Π_4 et Π_5 et laisserons les deux autres en Annexe A page 123. Après avoir obtenu ces résultats, nous en déduirons la $\mathcal{NP}$-complétude de tous les problèmes plus généraux.

Dans un deuxième temps nous nous intéresserons à la polynomialité des problèmes d'ordonnancement :

- $\Pi_6^P : 1|prec, (a_i = L_i = p, b_i = b) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$ (Section 5.2.6 page 73)
- $\Pi_7^P : 1|prec, (a_i = a, b_i = L_i = p) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$ (Section 5.2.7 page 80)
- $\Pi_8^P : 1|prec, (a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$ (Section 5.2.8 page 80)

Une fois que nous aurons prouvé la polynomialité de ces trois problèmes, nous en déduirons celle de tous les problèmes plus spécifiques. Intuitivement, les algorithmes appliqués pour résoudre ces problèmes sont tous basés sur la recherche d'un couplage maximum du fait de la structure particulière des tâches d'acquisition, $a_i = L_i = p$ et/ou $L_i = b_i = p$ et/ou $a_i = b_i = p, L_i = L$ dans tous les cas polynomiaux.

5.2.1 Étude de $\Pi_1^P : 1|prec, (a_i = b_i = p, L_i) \cup (\tau_i \geq 1), G_c\ complet|C_{max}$

Étudions le cas spécifique $\Pi_1^{P'} : 1|prec, (a_i = b_i = p, L_i) \cup (\tau_i = 1, pmtn), G_c\ complet|C_{max}$, où toutes les tâches de traitement ont un temps d'exécution égal à 1. Si ce cas est $\mathcal{NP}$-complet alors le cas général Π_1^P sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3.

Nous redonnons ici une définition particulière du problème 3-PARTITION à partir duquel nous réaliserons les réductions polynomiales (la première ci-dessous, et les deux autres en Annexe A page 123).

Données : Soit Q un ensemble de $3q$ éléments $Q = \{e_1, e_2, \ldots, e_{3q}\}$, et une borne $E \in \mathbb{Z}^+$, tels que $\forall i\ \frac{E}{4} < e_i < \frac{E}{2}$ et $\sum_{e_i \in Q} e_i = qE$.

Question : Est-ce que Q peut être partitionné en q ensembles disjoints $Q_1, Q_2, \ldots, Q_q$ tels que, pour $1 \leq i \leq q$, $\sum_{e_i \in Q_j} e_i = E$ (notons que chaque Q_i doit alors contenir exactement trois éléments de Q) ?

Problème de décision 5.2.1: 3-PARTITION [$SP15$]

Pour la suite nous pourrons supposer sans perte de généralité que $e_1 \leq e_2 \leq \ldots \leq e_{3q}$.

Théorème 5.2.1 :
Le problème de décider si une instance de notre problème $\Pi_1^{P'}$ possède un ordonnancement d'une certaine longueur y est $\mathcal{NP}$-complet.

Preuve :
Nous allons montrer que le problème $\mathcal{NP}$-complet 3-PARTITION se réduit vers le problème de décision $\Pi_1^{P'}$. À partir d'une instance de 3-PARTITION où $Q = \{e_1, e_2, \ldots, e_{3q}\}$, nous construisons l'instance suivante du problème de décision $\Pi_1^{P'}$. L'instance comprend $n = 4q$ tâches d'acquisition (resp. de traitement) telles que :

$$\begin{aligned} (a_i, L_i, b_i) = (2E, e_i, 2E) \ \text{ et } \ \tau_i = 1 \qquad & \text{pour } i = 1, \ldots, 3q, \\ (a_i, L_i, b_i) = (2E, 13E, 2E) \ \text{ et } \ \tau_i = 1 \qquad & \text{pour } i = 3q+1, \ldots, 4q. \end{aligned}$$

Existe t-il un ordonnancement valide avec $C_{max} \leq y = 17qE + 2$?

Dans la suite et pour les deux autres preuves suivantes basées sur la même structure, nous nommerons les tâches $A_1, A_2, \ldots, A_{3q}$ des **tâches de partition**, et $A_{3q+1}, \ldots, A_{4q}$ des **tâches de division**.

$\Rightarrow$ Nous allons montrer que s'il existe une solution au problème 3-Partition, alors $\Pi_1^{P'}$ admet une solution avec $C_{max} \leq y$. Nous supposons qu'il existe une solution au problème 3-PARTITION, ce qui entraîne qu'il existe des sous-ensembles Q_j pour $j = 1, \ldots, q$ avec $\sum_{i \in Q_j} e_i = E$. Commençons par ordonnancer les q tâches de division consécutivement sans chevauchement ni temps d'inactivité entre elles, toutes ces tâches sont équivalentes. Puis nous ordonnançons de manière récursive les tâches d'acquisition d'un sous-ensemble Q_j, où $\boldsymbol{Q_j = \{u, v, w\}}$, dans le temps d'inactivité d'une tâche de division, comme sur le schéma de la Figure 5.2.

Quant aux tâches de traitement, elles sont exécutées dans les slots d'inactivité de chaque tâche de partition de manière récursive, et les deux dernières tâches s'exécutent à la fin de

l'ordonnancement. Nous avons alors le makespan suivant :

$$C_{max} = \sum_{j=3q+1}^{4q} (p_{a_j} + L_j + p_{b_j}) + 2 = q(2E + 13E + 2E) + 2 = y$$

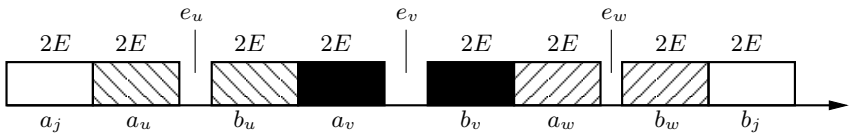


Fig. 5.2 – Ordonnancement d'un des q blocs de tâches avec un temps total d'inactivité égal à E

$\Leftarrow$ Nous allons montrer que l'existence d'un ordonnancement valide pour $\Pi_1^{P'}$ avec $C_{max} \leq y$, entraîne l'existence d'une solution au problème 3-PARTITION.

Tout d'abord, notons que la dernière tâche exécutée dans l'ordonnancement est forcément une tâche de traitement, nous allons donc analyser l'exécution des autres tâches avec un makespan de longueur $\boldsymbol{y - 1 = 17qE + 1}$.

Soit $\boldsymbol{\beta}$ la somme du temps séquentiel de toutes les tâches d'acquisition. Chacune des $4q$ tâches ont un temps séquentiel égal à $4E$, ce qui donne $\boldsymbol{\beta = 16qE}$. Donc nous n'avons le droit qu'à une durée d'inactivité de $qE + 1$ dans l'ordonnancement sans traitement. Or par définition, les slots d'inactivité créés par les tâches de partition sont tous trop petits pour qu'une sous-tâche d'acquisition puisse s'y exécuter (car $e_i < 2E$). Par conséquent, les tâches de partition ne peuvent être entrecroisées avec aucune autre tâche (mais elles peuvent par contre se trouver entièrement à l'intérieur d'une tâche de division). Et donc les slots d'inactivité des tâches de partition ne seront jamais utilisés dans l'ordonnancement sans traitement, nous sommes alors contraints de prendre en compte ces slots d'inactivité dans la durée d'inactivité totale permise dans l'ordonnancement sans traitement. Soit γ la durée totale des slots d'inactivité des tâches de partition qui est équivalent à :

$$\boldsymbol{\gamma = \sum_{i=1}^{3q} e_i = qE}$$

Au vu des valeurs de $\boldsymbol{\beta}$ et $\boldsymbol{\gamma}$, il ne nous reste plus qu'un seul temps d'inactivité que nous pouvons disposer hors des slots d'inactivité des tâches de partition (soit à l'intérieur d'une tâche de division, soit entre deux tâches).

Intéressons nous maintenant aux tâches de division. Nous allons montrer qu'il n'est pas possible de faire chevaucher l'exécution de plusieurs tâches de division. Nous allons étudier quatre cas :

- Pour une tâche de division D_i, nous allons essayer de combler son slot d'inactivité qu'avec des tâches de division. Chacune de ces tâches utilise $2E$ unités de temps par sous-tâche, donc en exécutant six tâches de division récursivement côte à côte, nous allons utiliser $6 * 2E$ du slot d'inactivité de D_i, laissant ainsi E unités de temps inutilisable. Ce qui n'est pas possible.
- Pour une tâche de division D_i, nous allons essayer de combler son slot d'inactivité avec une tâche de partition puis seulement avec des tâches de division. La tâche de partition P_j utilise $4E$ unités de temps séquentiel et laisse $e_j \leq \frac{E}{2} - 1$ temps d'inactivité inutilisables, donc il reste $9E - e_j$ unités de temps à utiliser dans D_i. L'exécution de quatre tâches de division récursivement côte à côte, utilise $4 * 2E$ du slot d'inactivité de D_i, laissant ainsi $E - e_j \leq \frac{E}{2} + 1$ unités de temps inutilisables. Ce qui n'est pas possible.

- Pour une tâche de division D_i, nous allons essayer de combler son slot d'inactivité avec deux tâches de partition puis seulement avec des tâches de division. Les deux tâches de partition P_j et P_k utilisent $8E$ unités de temps séquentiel et laissent $e_j + e_k \geq E - 2$ temps d'inactivité inutilisables, donc il reste $5E - e_j$ unités de temps à utiliser dans D_i. L'exécution de deux tâches de division récursivement côte à côte, utilise $2 * 2E$ du slot d'inactivité de D_i, laissant ainsi $E - e_j - e_k \geq 2$ unités de temps inutilisables. Ce qui n'est pas possible, car nous n'avons seulement droit qu'à une seule unité de temps d'inactivité.
- Pour une tâche de division D_i, nous allons essayer de combler son slot d'inactivité avec trois tâches de partition puis seulement avec des tâches de division. Les trois tâches de partition P_j, P_k et P_l utilisent $12E$ unités de temps séquentiel et laissent $e_j + e_k + e_l$ temps d'inactivité inutilisables, donc il reste $E - e_j - e_k - e_l$ unités de temps à utiliser dans D_i. Il est évident de voir qu'aucune sous-tâche de division ne peut être exécutée dans le résidu du slot de D_i.

Donc les tâches de division sont forcément exécutées côté à côte dans l'ordonnancement, et prennent $17qE$ unités de temps. Par conséquent toutes les tâches de partition sont exécutées dans les slots d'inactivité des tâches de division. Nous avons vu précédemment que seulement trois tâches de partition au maximum pouvaient être exécutées à l'intérieur d'une tâche de division.

Il reste à montrer que la somme des temps d'inactivité des trois tâches de partition par tâche de division est égal à E. Supposons que les ensembles Q_j ne donnent pas une solution pour le problème 3-PARTITION. Alors, il existe une tâche de division A_{3q+j} telle que $\sum_{\boldsymbol{A_i \in Q_j}} \boldsymbol{e_i} \geq \boldsymbol{(E+1)}$, et dont la somme des temps d'exécution des tâches de partition est égale à :

$$\begin{aligned}
\textstyle\sum_{A_i \in Q_j}(a_i + L_i + b_i) &= \textstyle\sum_{A_i \in Q_j}(4E + e_i) \\
&= 12E + \textstyle\sum_{A_i \in Q_j} e_i \\
&\geq 12E + (E+1) \\
&> 13E = L_{3q+j}
\end{aligned}$$

Donc les tâches de partition de Q_j ne peuvent pas être exécutées dans le slot d'inactivité de longueur L_{3q+j}, ce qui montre que $\sum_{i \in Q_j} e_i \leq E$ pour $j = 1, \ldots, q$. De plus, nous savons que s'il existe un ensemble Q_k tel que $\sum_{A_i \in Q_k} e_i < E$, alors il existe un ensemble Q_l tel que $\sum_{A_i \in Q_l} e_i > E$. Ce qui entraîne que $\sum_{A_i \in Q_j} e_i = E$ pour $j = 1, \ldots, q$, et par conséquent $Q_1, \ldots, Q_q$ représentent une 3-PARTITION.

Nous avons donc 3-PARTITION $\leq_T \Pi_1^{P'}$, et nous savons que 3-PARTITION (2.2.9) est $\mathcal{NP}$-complet. Ainsi, d'après le Lemme 2.1.2 nous pouvons conclure que le problème $\Pi_1^{P'}$ est $\mathcal{NP}$-complet.

□

Nous discuterons de l'étude de l'approximation du problème Π_1^P à la Section 5.3.1 page 81.

5.2.2 Étude du problème $\Pi_2^P : 1|prec, (a_i = b_i = L_i) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$

Étudions le cas spécifique $\Pi_2^{P'} : 1|prec, (a_i = b_i = L_i) \cup (\tau_i = 1), G_c\ complet|C_{max}$, où toutes les tâches de traitement ont un temps d'exécution égale à 1. Si ce cas est $\mathcal{NP}$-complet alors le cas général Π_2^P sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3.

Théorème 5.2.2 :
Le problème de décider si une instance de notre problème $\Pi_2^{P'}$ possède un ordonnancement optimal d'une certaine longueur y est $\mathcal{NP}$-complet.

Preuve :
La preuve est donné en Annexe A à la page 123.
Nous discuterons de l'étude de l'approximation du problème Π_2^P à la Section 5.3.1 page 81.

5.2.3 Étude du problème $\Pi_3^P : 1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$

Étudions le cas spécifique $\Pi_3^{P'} : 1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i = 1), G_c\ complet|C_{max}$, où toutes les tâches de traitement ont un temps d'exécution égale à 1. Si ce cas est $\mathcal{NP}$-complet alors le cas général Π_3^P sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3.

Théorème 5.2.3 :
Le problème de décider si une instance de notre problème $\Pi_3^{P'}$ possède un ordonnancement optimal d'une certaine longueur y est $\mathcal{NP}$-complet.

Preuve :
La preuve est donnée en Annexe A.2 à la page 125.
Nous discuterons de l'étude de l'approximation du problème Π_3^P à la Section 5.3.1 page 81.

5.2.4 Étude du problème $\Pi_4^P : 1|(a_i = a, L_i = L, b_i) \cup (\tau_i, pmnt), G_c\ complet|C_{max}$

Théorème 5.2.4 :
Le problème de décision associé à Π_4^P est $\mathcal{NP}$-complet.

Preuve :
D'après le Théorème 4.2.6 à la page 50, nous pouvons déterminer la complexité d'un problème lorsque nous connaissons celle du problème symétrique, ceci reste vrai avec les tâches de traitement. Dans la figure 3.5 représentant les résultats d'Orman et Potts, nous avons une symétrie entre $\mathbf{1|a_i = a, L_i = L, b_i|C_{max}}$ et $\mathbf{1|a_i, L_i = L, b_i = b|C_{max}}$. En rajoutant la contrainte de précédence avec des tâches de traitement, les deux problèmes restent symétriques, donc le problème Π_3^P est symétrique au problème Π_4^P.

En utilisant le Théorème 4.2.6, nous en déduisons que le problème de décision associé à Π_4^P est $\mathcal{NP}$-complet tout comme l'est le problème Π_3^P.

□

Nous discuterons de l'étude de l'approximation du problème Π_4^P à la Section 5.3.1 page 81.

5.2.5 Étude du problème $\Pi_5^P : 1|(a_i = a, L_i, b_i = b) \cup (\tau_i, pmnt), G_c\ complet|C_{max}$

Pour montrer la $\mathcal{NP}$-complétude de ce problème, nous allons utiliser le résultat de complexité du problème Π_1^P. En faisant varier les valeurs des paramètres fondamentaux a_i et b_i, nous pouvons proposer la remarque suivante :

Remarque 5.2.1 :
Le problème $\Pi_5^P : 1|(a_i = a, L_i, b_i = b) \cup (\tau_i, pmnt), G_c\ complet, |C_{max}$ est une généralisation du problème $\Pi_1^P : 1|(a_i = b_i = p, L_i) \cup (\tau_i, pmnt), G_c\ complet, |C_{max}$. En effet, en posant $a = b = p$ sur une instance de Π_5^P nous obtenons une instance de Π_1^P.

Théorème 5.2.5 :
Le problème de décision associé à Π_5^P *est* $\mathcal{NP}$*-complet.*

Preuve :
La Remarque 5.2.1 implique la $\mathcal{NP}$-complétude du problème Π_5^P d'après le Lemme 2.1.3 de la Section 2.1.4.

□

Nous discuterons de l'étude de l'approximation du problème Π_5^P à la Section 5.3.1 page 81.

5.2.6 Étude du problème $\Pi_6^P : 1|(a_i = L_i = p, b_i) \cup (\tau_i, pmnt), G_c \ complet|C_{max}$

Ce problème est composé de n tâches d'acquisition toutes sur le même modèle $a_i = L_i = p, b_i$. La première sous-tâche et le temps d'inactivité sont fixés à la même constante p, $p \in \mathbb{N}^*$. La seconde tâche peut prendre n'importe quelle valeur.

Nous reprenons le modèle utilisé dans la Section 4.2.3 page 47, tel que l'ensemble des n tâches d'acquisition contient deux sous-ensembles de tâches : le premier sous-ensemble noté $\boldsymbol{K}$ est composé de toutes les tâches d'acquisition A_i telles que $b_i \leq p$, le second sous-ensemble noté $\boldsymbol{S}$ est composé par tous les autres tâches. Deux tâches S_i et S_j dans S ne peuvent pas être emboîtées l'une dans l'autre, donc $\{i, j\} \notin G_c$ et nous enlevons automatiquement ces arêtes. Pour cette section, nous dirons que G_c est un graphe complet lorsque K formera une clique, S un stable et $\forall x \in K, \forall y \in S,\ \{x, y\} \in G_c$.

Soit un ordonnancement quelconque. On déduit un couplage M de G_c de la façon suivante :
Si A_i et A_j sont emboîtées, alors $\{i, j\} \in M$.

Remarque : Si a_i est ordonnancée avant a_j, on dit que A_i (resp. A_j) est la tâche première (resp. seconde) du couple. Remarquons qu'une tâche première appartient toujours à K.

Les arêtes de M sont de deux types selon les tâches secondes : $K - K$ si les deux extrémités sont dans K, et $K - S$ sinon.

Les tâches non saturées par M sont appelées tâches isolées.

Dans la suite, nous allons donner deux Remarques (5.2.2, 5.2.3) et une Propriété (5.2.1) par rapport à M que l'on peut supposer sans perte de généralités pour un ordonnancement optimal et un $\boldsymbol{G_c}$ **quelconque**. Pour simplifier, nous utiliserons les notations suivantes :

- $\forall A_i \in K$, les tâches d'acquisition sont notées $A_i = (a_i = L_i = p, b_i)$, et les tâches de traitement associées τ_i^K.
- $\forall A_i \in S$, les tâches d'acquisition sont notées $A_i = (c_i = L_i = p, d_i)$, et les tâches de traitement associées τ_i^S.

Remarque 5.2.2 *Sans perte de généralité, on peut supposer que si* $\tau_i^S > \tau_j^S$ *alors* S_i *est exécutée avant* S_j*. En effet, si ce n'est pas le cas, nous avons* $\tau_i^S < \tau_j^S$*. Il est facile de voir que la conservation du même ordonnancement en échangeant* S_i *et* S_j *ne peut que réduire la durée de l'ordonnancement :*

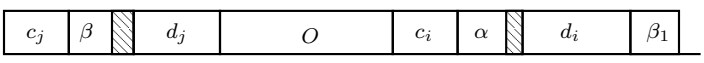


Fig. 5.3 – Première configuration

En échangeant :

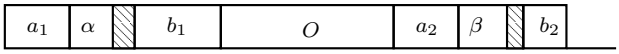
Fig. 5.4 – Deuxième configuration

Sur la première configuration (Figure 5.3), les éventuelles tâches de traitement exécutées pendant O et α sont les mêmes que dans O' et α' sur la seconde configuration (Figure 5.4) à l'exception des tâches de traitement de S_i et S_j. L'échange entre S_j et S_i entraîne que $\alpha' \geq \alpha$, et ainsi β_2 contient autant ou moins de tâches de traitement à exécuter que β_1, donc durée$(\beta_2) \leq$ durée (β_1).

Remarque 5.2.3 *De la même manière que pour la Remarque 5.2.2, nous pouvons supposer sans perte de généralités que si K_i et S_j sont des tâches secondes, et si $\tau_i^K > \tau_j^S$ (resp. $\tau_j^S > \tau_i^K$) alors K_i est exécutée avant S_j (resp. S_j avant K_i) comme tâche seconde.*

Proposition 5.2.1 *Le couplage M est maximal pour un G_c quelconque.*

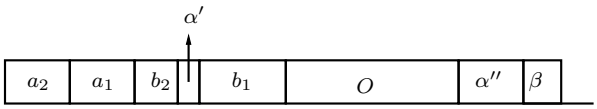
Preuve :
Supposons que ce n'est pas le cas, nous avons donc deux tâches isolées A_1 et A_2 qui sont reliées dans G_c (remarque : $min(b_1, b_2) \leq p$) :



avec A_1-A_2
(remarque : $min(b_1, b_2) \leq p$)

Fig. 5.5 – Première configuration

À partir de la première configuration (Figure 5.5), nous obtenons comme temps séquentiel $\boldsymbol{C_{max}(\text{cas1}) = 4p + |O| + b_1 + b_2}$. Regardons ce que donne l'emboîtement de A_1 avec A_2 (en supposant sans perte de généralité que $b_2 \leq p$) :



avec $\alpha = \alpha' + \alpha''$

Fig. 5.6 – Deuxième configuration

À partir de la deuxième configuration (Figure 5.6), nous obtenons comme temps séquentiel $\boldsymbol{C_{max}(\text{cas2}) = 3p + |O| + b_1 + |\alpha''| + |\beta|}$.

Soit $D = C_{max}(\text{cas1}) - C_{max}(\text{cas2}) = p + b_2 - \beta - \alpha''$. Nous avons deux cas à étudier :

1. cas $\alpha'' = 0 \Rightarrow \alpha = \alpha'$: Comme $\beta \leq p$ donc $D \geq 0$
2. cas $\alpha' = p - b_2 \Rightarrow \alpha'' = \alpha - \alpha' = \alpha - p + b_2$

Nous obtenons ainsi $D = p + b_2 - \beta - \alpha + p - b_2 = 2p - \beta - \alpha$, or $\alpha \leq p$ et $\beta \leq p$, donc $D \geq 0$. Nous en concluons que le second ordonnancement est toujours meilleur.

□

Prenons maintenant en compte le fait que $\boldsymbol{G_c}$ **est complet**. Nous allons donner plusieurs résultats permettant de conclure sur le fait qu'un couplage maximum M permet d'obtenir un ordonnancement optimal.

Remarque 5.2.4 *Soit K_i et K_j deux tâches premières avec $b_i+\tau_i^K > b_j+\tau_j^K$. On peut toujours supposer que la tâche K_i est exécutée avant K_j. En effet, si ce n'est pas le cas, l'échange des deux tâches ne peut que réduire la durée de l'ordonnancement.*

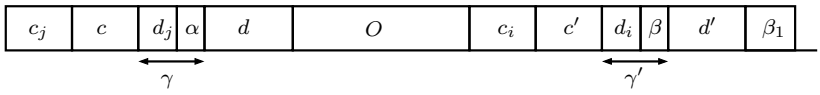


Fig. 5.7 – Première configuration

En échangeant :

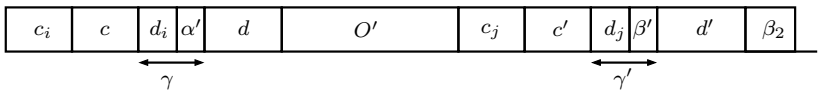
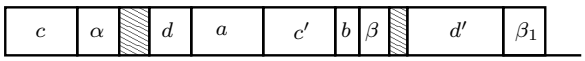


Fig. 5.8 – Deuxième configuration

Toutes les sous-tâches d'acquisition seconde de K_i et K_j, et la tâche de traitement K_j dans γ, O, γ' peuvent être remplacées par les sous-tâches d'acquisition seconde de K_i et K_j, et la tâche de traitement de K_i. Ainsi $\beta_2 \leq \beta_1$ (moins de tâches restantes à exécuter).

Proposition 5.2.2 *Nous pouvons toujours supposer que les tâches isolées de S peuvent être exécutées à la fin de l'ordonnancement.*

Preuve :
En effet, si ce n'est pas le cas, on a donc une tâche S_1 suivi d'un couple $K - S_2$ par exemple. Soit τ_1^S (resp. τ_2^S) la durée d'exécution de la tâche de traitement associé à S_1 (resp. S_2) :



$\tau_1^S \geq \tau_2^S$ d'après la Remarque 5.2.2

Fig. 5.9 – Première configuration

Nous obtenons comme temps séquentiel $\boldsymbol{C_{max}(\text{cas1}) = 2p + d + 3p + d' + \beta_1}$. En échangeant l'arête $K - S_2$ par $K - S_1$, nous obtenons :

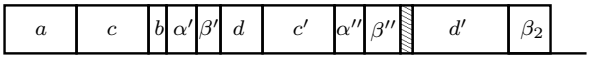


Fig. 5.10 – Deuxième configuration où $\alpha' = min(p - b, \alpha)$ (resp. $\beta' = min(p - b, \beta)$) est le temps séquentiel exécuté dans le premier slot, et $\alpha'' = \alpha - \alpha'$ (resp. $\beta'' = \beta - \beta'$) le temps séquentiel résiduel de ce qui n'a pas été exécuté dans le premier slot

Nous obtenons comme temps séquentiel $\boldsymbol{C_{max}(\text{cas2}) = 3p + d + 2p + d' + \beta_2}$. Montrons que $\beta_2 \leq \beta_1$, nous étudions deux cas :

1. Si $\alpha'' = 0 : \beta' = 0,\ \beta'' = \beta$ et donc $\beta_2 = \tau_2^S$. Or comme $\beta_1 \geq \tau_2^S$, on a $\beta_2 \leq \beta_1$.
2. Sinon : $\alpha' = p - b \Rightarrow \beta' = 0, \alpha'' = \alpha - \alpha' = (\alpha - p) + b \Rightarrow \alpha'' \leq b$
   $\Rightarrow \beta'' \geq \beta \Rightarrow \beta_2 \leq \beta_1$.

Donc la seconde configuration (Figure **??**) est toujours meilleure.

□

Lemme 5.2.1 *Si G_c est complet (où nous avons enlevé les arêtes impossibles), alors nous pouvons supposer que le couplage associé à un ordonnancement optimal est maximum.*

Preuve :
Prenons un couplage maximal qui donne un ordonnancement optimal. Supposons que ce couplage n'est pas de taille maximum, dans ce cas deux tâches isolées sont exécutées. Ces tâches sont forcément de S (couplage maximal), et deux tâches de K au moins sont couplées entre elles (couplage de taille non maximum). D'après les résultats précédents, nous pouvons supposer sans perte de généralités qu'entre ces deux tâches isolées de S et le couple $K - K$ le plus proche de ces tâches, il y a un bloc noté O composé de k arêtes de type $K - S$, $k \geq 0$ (voir Figure 5.11). Soient α_i les slots créés par les $k + 1$ couples et les deux tâches isolées numérotés de 1 à $k + 3$.

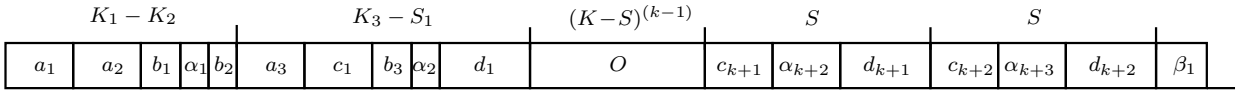


Fig. 5.11 – Première configuration

Nous avons $\boldsymbol{C_{max}(\text{cas1}) = \mathbf{T} + p + b_2 + \beta_1}$, où $\mathbf{T} = 3(k+2)p + \sum_{i=1}^{k+2} d_i$

Transformation de la première configuration ($(K-S)^{k+1}\ S\ S\ \beta_2$) à la deuxième configuration ($(K–S)^{k+2}\ \beta_2$). D'après les hypothèses, l'ordonnancement obtenu ne doit pas être meilleur.

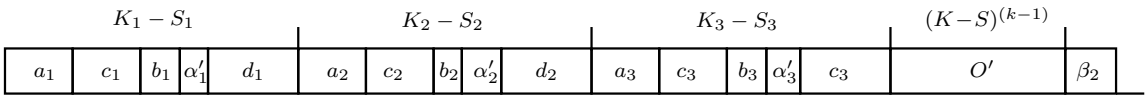


Fig. 5.12 – Deuxième configuration

Soient α'_i les slots créés par les $k + 2$ couples numérotés de 1 à $k + 2$. Nous avons le temps séquentiel $\boldsymbol{C_{max}(\text{cas2}) = \mathbf{T} + \beta_2}$. Et donc C_{max}(cas2)− C_{max}(cas1)= $\beta_2 - p - b_2 - \beta_1$. De plus, remarquons que $\alpha_1 = \alpha'_1$, $\beta_1 \geq \tau_{k+2}^S$

Nous pouvons supposer d'après les propriétés précédentes :
– $b_1 + \tau_1^K \geq b_2 + \tau_2^K \geq b_3 + \tau_3^K \geq \ldots \geq b_{k+2} + \tau_{k+2}^K$
– $\tau_1^S \geq \tau_2^S \geq \tau_3^S \geq \ldots \geq \tau_{k+2}^S$

Nous avons plusieurs cas possibles. Supposons :

1. qu'il existe un temps d'inactivité dans α'_i ($i > 2$), donc $\exists i > 2$ tel que $\alpha'_i < p - b_i$
   $\Rightarrow b_i + \tau_i^K + \tau_{i-1}^S < p \Rightarrow \beta_2 = \tau_{k+2}^S$ car $\forall j > i$, $b_j + \tau_j^K + \tau_{j-1}^S < p$ donc aucun slot ne sera comblé $\Rightarrow \beta_2 \leq \beta_1 \Rightarrow C_{max}$(cas2)$\leq C_{max}$(cas1)

2. $\forall i > 2\ \alpha'_i = p - b_i$ et $\alpha'_2 = p - b_2$ alors la transformation a moins de slot ou autant que l'ordonnancement d'origine C_{max}(cas2)$\leq C_{max}$(cas1). En effet dans la deuxième configuration il y a un taux d'occupation de 100% du processeur à partir de l'instant $3p$ (la seule inactivité du processeur dans la deuxième configuration a lieu si $b_1 + \alpha_1 < p$, mais ce temps d'inactivité a également lieu dans la première configuration).

3. qu'il n'existe pas de slot excepté en α'_2, $\forall i > 2\ \alpha'_i = p - b_i$ et $\alpha'_2 < p - b_2$, alors $\beta_2 \leq \beta_1 + p - (b_2 + \alpha'_2) \leq \beta_1 + p \Rightarrow C_{max}$(cas2)$\leq C_{max}$(cas1)

Dans les trois cas, l'ordonnancement de la deuxième configuration est aussi bon ou meilleur que celui de la première. Donc nous pouvons supposer qu'à tout ordre optimal, on peut en déduire un couplage de taille maximum.

□

Les lemmes, propriétés et remarques que nous venons de donner nous permettent de définir un algorithme fournissant en temps polynomial une solution optimale au problème Π_6^P. En effet le Lemme 5.2.1 nous apporte le fait que notre solution optimale est équivalente à trouver un couplage maximum, et d'après les propriétés précédentes nous pouvons supposer que pour le couplage maximum donnant une solution optimale, nous avons les relations d'ordre suivantes :

- $b_1 + \tau_1^K \geq b_2 + \tau_2^K \geq b_3 + \tau_3^K \geq \ldots \geq b_{k+2} + \tau_{k+2}^K$
- $\tau_1^S \geq \tau_2^S \geq \tau_3^S \geq \ldots \geq \tau_{k+2}^S$

Il suffit alors de voir s'il vaut mieux coupler deux éléments de K ensemble plutôt qu'un élément de K avec un élément de S. Nous décomposons le problème en trois cas :

- Si $|K| = |S|$, nous trions l'ensemble des tâches d'acquisition A_i de K (resp. S) par ordre décroissant selon $b_i + \tau_i$ (resp. τ_i). La solution consiste à coupler le plus grand élément de K avec celui de S à chaque fois.

- Si $|K| < |S|$, même solution que précédemment, mais lorsque K est vide, il suffit d'exécuter à la chaîne les éléments restants de S dans l'ordre.

- Si $|K| > |S|$:

Supposons dans ce qui suit que $|V| = K + S = 2n$. D'après le Lemme 5.2.1, il existe un couplage maximum (parfait dans ce cas là) qui est associé à un ordonnancement optimal. De même étant donné l'ensemble P des n tâches premières, il est facile de construire un des meilleurs ordonnancements associés.

L'algorithme consisterait à trier les n tâches premières par rapport à $b_i + \tau_i$ décroissant, puis à trier les n tâches secondes D par rapport à τ_i. Ainsi nous obtenons $P_1, P_2, \ldots, P_n$ (resp. $D_1, D_2, \ldots, D_n$)) les tâches ordonnancées de P (resp de D). Nous ordonnançons $\{P_1, D_1\}$, puis $\{P_2, D_2\}$ etc ...

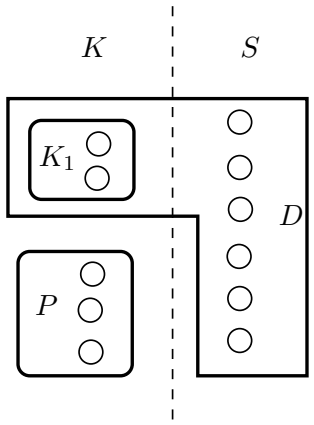


Fig. 5.13 – Illustration du découpage des sommets de G_c

On en déduit la durée de l'ordonnancement selon le type d'ordonnancement obtenu :

1. S'il y a un temps d'inactivité après le premier slot, la durée totale est $3pn + \sum_{A_i \in D} b_i + \tau'_n$, où τ'_n est la durée de la tâche de traitement de la tâche D_n.
2. S'il y a un temps d'inactivité qu'au premier slot, la durée de l'ordonnancement est $\sum_{A_i \in D \cup P}(b_i + \tau_i) + (2n+1)p - (b_1 + \tau_1)$, où $(b_1 + \tau_1)$ concerne P_1
3. Il n'y a aucun temps d'inactivité : l'algorithme est optimal et la durée de l'ordonnancement est $\sum_{A_i \in D \cup P}(b_i + \tau_i) + 2pn$.

Nous avons forcément $S \subseteq D$. Notons K_2 l'ensemble des $n - |S|$ tâches de K qui minimise b_i. Prenons $D = S \cup K_2$ et $P = K - K_2$. Nous allons montrer qu'à partir de ce simple algorithme de trie et de couplage nous pouvons obtenir en temps polynomial une solution optimal à notre ordonnancement. Commençons par remarquer qu' à la vue des trois types d'ordonnancement , seulement trois points sont importants : $min\{\sum_{A_i \in D} b_i\}$, $(b_1 + \tau_1)$ et τ'_n.

Cet algorithme permet d'obtenir un ordonnancement avec $min\{\sum_{A_i \in D} b_i\}$, d'après le type d'ordonnancement 1, si la tâche avec le τ_i minimum est dans D et qu'il y a un slot d'inactivité après le slot 1 c'est fini, l'algorithme est forcément optimal. En effet $\sum_{A_i \in D} b_i$ est minimal sur l'ensemble des D possible et τ'_n également.

Dans le cas contraire, si la tâche avec le τ_i minimum est dans P et s'il y a un instant d'inactivité de durée $\gamma > 0$ sur le slot n. La seule possibilité pour améliorer l'ordonnancement est de diminuer τ'_n. Soit X l'ensemble des tâches de P telles que $\tau_i \leq \tau'_n$. Prenons la tâche A_i qui maximise b_i dans K_2, et $\forall A_j \in X$ échangeons A_i avec A_j. Gardons le meilleur ordonnancement associé (ça peut être celui que nous avions déjà). Remarquons que cet ordonnancement permet d'avoir $min\{\sum_{A_i \in D} b_i + \tau_i\}$.

Si le meilleur ordonnancement a un temps d'inactivité $\gamma > 0$ sur le slot n alors l'ordonnancement est optimal, sinon nous sommes peut être dans le type d'ordonnancement 2, et la seule façon d'améliorer l'ordonnancement serait de diminuer l'inactivité du premier slot. Soit Y l'ensemble des tâches de K_2 tel que $\tau'_i + b_i > \tau_1 + b_1$ et A_k la tâche de P qui minimise b_i. $\forall A_j \in Y$ échangeons A_k avec A_j. Gardons le meilleur ordonnancement associé (ça peut être celui que nous avions déjà), c'est forcément l'optimal.

Cette méthode permettant d'obtenir un ordonnancement optimal est donné par l'algorithme 7, qui retourne une solution optimale au problème Π_6^P. Il utilise l'algorithme $COUPLER$ qui couple les éléments de P avec ceux de S par ordre décroissant. Cet algorithme dépend essentiellement du couplage du début, l'algorithme donne donc une solution optimale en temps $O(nlog(n))$.

Algorithme 7 : Algorithme optimal pour le problème Π_6^P

Données : P, D, K, S, K_2
Résultat : C_{max}^{opt}
début
  Trier les éléments de P par ordre décroissant de $b_i + \tau_i$
  Trier les éléments de D par ordre décroissant de τ_i
  $COUPLER(P, D)$
  si *il y a un trou au slot n* **alors**
    si $\tau'_n \in D$ **alors**
      Fin de l'algorithme
    sinon
      Soit $X \subseteq P$ tel que $K_i \in X$ si $\tau_i^k < \tau'_n$
      Soit $A_i \in K_2$ qui maximise b_i
      Chercher si possible un $A_j \in X$ tel que $COUPLER(P\backslash A_j + A_i, D\backslash A_i + A_j)$ donne un meilleur ordonnancement
      si *il y a un trou au slot n* **alors**
        Fin de l'algorithme
  Soit $Y \subseteq K_2$ tel que $K_i \in Y$ si $b_i + \tau_i > b_1 + \tau'_1$
  Soit $A_i \in P$ qui minimise b_i
  Chercher si possible un $A_j \in Y$ tel que $COUPLER(P\backslash A_j + A_i, D\backslash A_i + A_j)$ donne un meilleur ordonnancement
fin

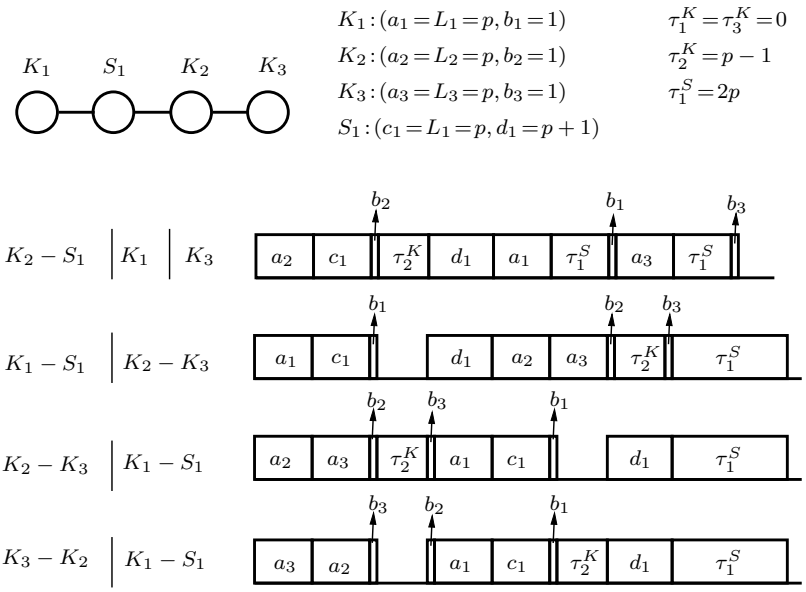


Fig. 5.14 – Contre-exemple

Remarque 5.2.5 *Les preuves que nous venons de donner sont basées sur l'échange classique en ordonnancement de deux tâches comme le fait Smith [64]. Intuitivement avec un graphe de compatibilité quelconque, un couplage maximum devrait induire un ordonnancement optimal, mais ce n'est pas le cas comme nous pouvons l'observer sur le contre-exemple de la Figure 5.14. Sur celui-ci les ordonnancements optimaux donnés ne sont pas obtenus avec un couplage de taille maximum (contrairement au cas G_c complet). Donc cette propriété n'est pas conservée si G_c est non complet.*

5.2.7 Étude du problème $\Pi_7^P : 1|(a_i, L_i = b_i = p) \cup (\tau_i, pmnt), G_c \ complet|C_{max}$

Théorème 5.2.6 :
Le problème de décision Π_7^P est polynomial.

Preuve :
D'après le Théorème 4.2.6 de la page 50, nous pouvons déterminer la complexité d'un problème lorsque nous connaissons celle du problème symétrique, ceci reste vrai avec les tâches de traitement. Dans la figure 3.5 représentant les résultats d'Orman et Potts, nous avons une symétrie entre $\mathbf{1|a_i = L_i = p, b_i|C_{max}}$ et $\mathbf{1|a_i, L_i = b_i = p|C_{max}}$. En rajoutant la contrainte de précédence avec des tâches de traitement, les deux problèmes restent symétriques, donc le problème Π_6^P est symétrique au problème Π_7^P.

En utilisant le Théorème 4.2.6, nous en déduisons que le problème Π_7^P est polynomial tout comme l'est le problème Π_6^P.

□

5.2.8 Étude du problème $\Pi_8^P : 1|(a_i = b_i = p, L_i = L) \cup (\tau_i, pmnt), G_c \ complet|C_{max}$

Théorème 5.2.7 :
Le problème de décision Π_8^P est polynomial.

Preuve :
La preuve de ce problème est similaire à celle pour Π_6^P, en effet il y a deux cas selon les valeurs de L et p. Si $p > L$, les tâches d'acquisition sont exécutées les unes après les autres de manière décroissante selon le temps d'exécution de leurs tâches de traitement associées. Si $p \leq L$, nous exécutons les tâches d'acquisition le plus tôt possible au premier slot libre de manière décroissante selon le temps d'exécution de leurs tâches de traitement associées.

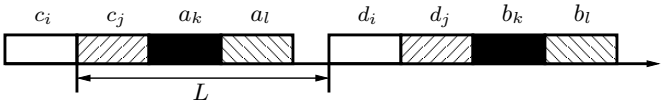


Fig. 5.15 – Un bloc de tâches ordonnancées de manière contiguë pour Π_8

□

5.2.9 Vision globale de la complexité

Nous avons montré la $\mathcal{NP}$-complétude des cinq problèmes Π_1^P, Π_2^P, Π_3^P, Π_4^P, et Π_5^P, et la polynomialité des trois problèmes Π_6^P, Π_7^P et Π_8^P. À partir de ces résultats nous en déduisons la $\mathcal{NP}$-complétude de tous les problèmes plus généraux que les cinq cas qui ont été étudiés

précédemment, et la polynomialité de tous les problèmes plus spécifique que les trois cas étudiés. En effet à partir des résultats de complexité de Π_i^P, pour $i = 1, \ldots, 8$, et d'après le Lemme 2.1.3, la polynomialité de tous les problèmes plus généraux (resp. spécifiques) que Π_i, pour $i = 1, \ldots, 5$ (resp. pour $i = 6, \ldots, 8$), sont également $\mathcal{NP}$-complets (resp. polynomiaux). Ceci finit l'étude globale des problèmes d'ordonnancement avec tâches d'acquisition en présence d'un graphe de précédence (voir Figure 5.1). Dans la section suivante, nous allons continuer l'analyse des problèmes Π_i, pour $i = 1, \ldots, 5$, en cherchant les meilleures bornes d'approximation possibles.

5.3 Étude de l'approximation

5.3.1 Algorithme d'approximation avec garantie de performance

L'approximation pour les problèmes étudiés dans ce chapitre n'a pas été étudiée comme précédemment. Comme il en a été question pour la complexité de ces problèmes, l'introduction des tâches de traitement ne change pas la complexité puisque les résultats obtenus par Orman et Potts (représentés à la Figure 3.5) sont équivalents lorsque les tâches de traitement ont toutes une durée d'exécution égale à zéro. Donc lorsque les tâches de traitement sont toutes de longueur zéro, les résultats d'approximation sont équivalents à ceux présentés dans l'état de l'art à la Section 3.3.2.0 page 34. Il n'y a pas eu énormément de travaux portant sur l'approximation de ces problèmes en particulier, nous redonnons dans le Tableau 5.1 les résultats les plus importants réalisés par Ageev, Baburin et Kononov [26, 1].

Problème	Ratio d'approx.	Borne de non approx.	Complexité	Référence
$a_i = b_i = 1, L_i$	$\frac{7}{4}$	?	$O(n\ log\ n)$	[26]
a_i, L_i, b_i	$\frac{7}{2}$	$2 - \epsilon$	$O(n\ log\ n)$	[1]
$a_i < b_i, L_i$	3	$2 - \epsilon$	$O(n\ log\ n)$	[1]
$a_i > b_i, L_i$	3	$2 - \epsilon$	$O(n\ log\ n)$	[1]
$a_i = b_i, L_i$	$\frac{5}{2}$	$2 - \epsilon$	$O(n\ log\ n)$	[1]

TAB. 5.1 – Résultats d'approximation et non-approximabilité donnés par Ageev, Baburin et Kononov

Il pourrait être intéressant d'étudier l'approximation de ces problèmes lorsque les tâches de traitement sont toutes égales à une constante τ, et d'avoir une courbe représentant l'évolution de la performance relative ρ pour les problèmes étudiées en fonction de τ.

Nous savons que pour $\tau = 0$, nous avons le ρ le plus grand (car pire des cas). Au vu des résultats présentés dans le tableau 5.1, le plus grand ρ possible sera égal à $\frac{7}{2}$. Pour un τ très grand, les temps d'inactivité créés par les tâches d'acquisition seront toujours comblés à l'exception du premier. Mais puisque nous avons G_c qui est complet, il suffit de faire chevaucher les tâches de la meilleure des manières pour que nous obtenions l'optimal $\rho = 1$. Mais selon certains problèmes cela peut être très difficile à trouver, nous proposons donc la conjecture 5.3.1 sur la variation de ρ et sur l'optimal atteint.

Conjecture 5.3.1 *La courbe de la variation de la performance relative ρ selon τ pour tout problème étudié dans ce chapitre variera de cette manière entre au plus $\frac{7}{2}$ et l'optimal 1 qui est atteint.*

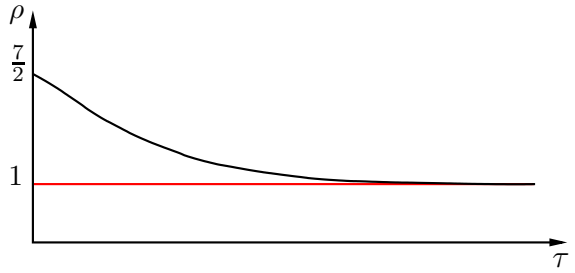


Fig. 5.16 – Variation de ρ selon la valeur de τ

5.4 Conclusion

Dans ce chapitre, nous avons cette fois-ci étudié les problèmes d'ordonnancement avec tâches d'acquisition en présence d'un graphe complet de compatibilité G_c et en introduisant les tâches de traitement. Nous étudions les mêmes problèmes que dans le **Chapitre 4** afin de mieux apprécier l'impact de cette nouvelle contrainte, tout en revenant à un graphe de compatibilité complet (comme pour Orman et Potts).

Le but recherché tout au long du chapitre a été de déterminer l'impact de l'introduction des tâches de traitement sur ces problèmes, et analyser les cas critiques se trouvant à la frontière entre la polynomialité et la $\mathcal{NP}$-complétude selon la valeurs des paramètres, et puis comparer la complexité obtenue par rapport à Orman et Potts et par rapport au Chapitre 4.

Le chapitre est découpé en deux parties, la première portant sur l'étude de la complexité des problèmes critiques et par conséquent tous les problèmes plus généraux, et la deuxième partie portant sur l'étude de l'approximation de ces mêmes problèmes.

Nous donnons cinq preuves de $\mathcal{NP}$-complétude pour les problèmes notés Π_i^P pour $i = 1, \ldots, 5$, et trois preuves de polynomialité pour les problèmes Π_6^P, Π_7^P et Π_8^P (Voir Figure 4.17). Les preuves, cette fois-ci, ne sont pas basées sur le graphe de compatibilité G_c (qui est complet), mais plutôt sur des réductions à partir de problèmes connus de partitionnement d'ensemble (3-PARTITION, PARTITION). À partir de ces résultats nous en déduisons la $\mathcal{NP}$-complétude de tous les problèmes plus généraux.

Pour ce qui est des preuves de polynomialité des problèmes notés Π_6^P, Π_7^P et Π_8^P, les algorithmes donnés sont toujours basés sur des couplages de taille maximum mais dépendent fortement de l'ordre des tâches de traitement associé aux tâches d'acquisition selon leur temps d'exécution. À partir de ces résultats nous en déduisons la polynomialité de tous les problèmes plus spécifiques. La visualisation globale de la complexité des problèmes est représentée sur la Figure 5.17, nous avons mis en avant l'évolution de la complexité des problèmes lors de l'introduction des tâches de traitement.

Il est intéressant d'observer que la complexité des problèmes ne change pas avec l'introduction de ces nouvelles tâches, les preuves deviennent plus difficiles mais restent sur le même schéma. Le plus gros changement a lieu sur les problèmes polynomiaux, les tâches de traitement rendant les problèmes plus difficiles, l'étude devient délicate mais la complexité reste néanmoins polynomiale. Tout dépend de l'ordre d'ordonnancement des tâches que nous couplons selon les valeurs du temps d'exécution des tâches de traitement.

La deuxième partie sur l'approximation n'a pas fait preuve d'une étude approfondie de notre part, puisque dans le pire des cas où les tâches de traitement sont de longueur nulle,

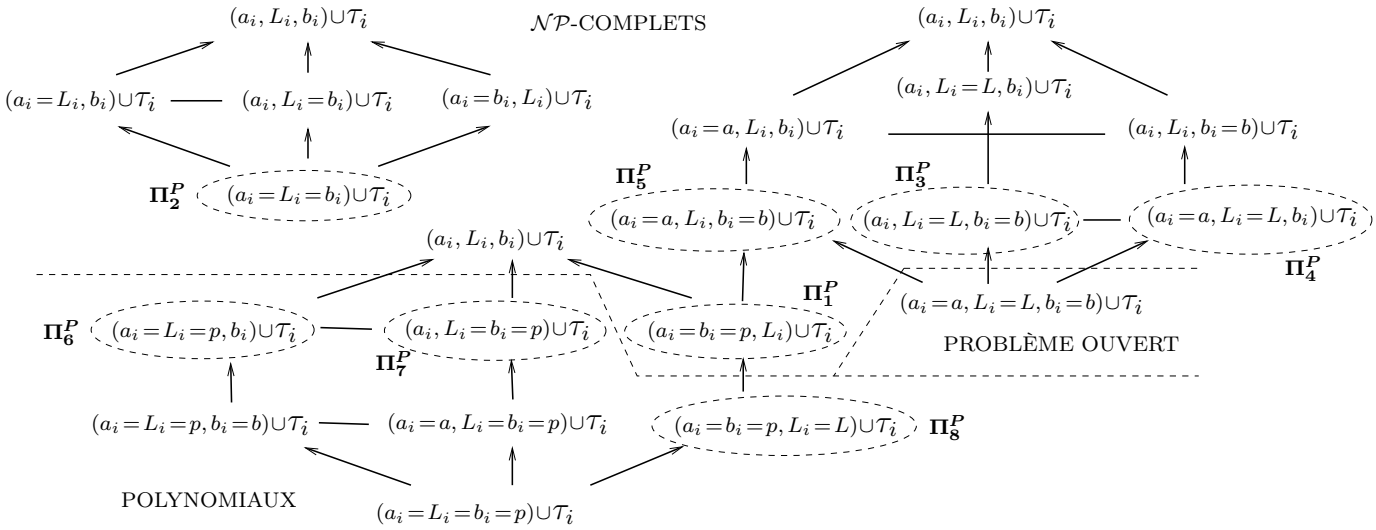


Fig. 5.17 – Visualisation globale de l'impact de l'introduction des tâches de traitement sur la complexité des problèmes d'ordonnancement avec tâches d'acquisition sur mono processeur. La complexité est la même que celle d'Orman et Potts.

nous revenons aux problèmes classiques déjà étudiés dans la littérature. Soulignons que l'étude d'approximation lorsque toutes les tâches de traitement sont égales à une constante peut être intéressante à faire, montrant ainsi l'évolution du ratio d'approximation pour chaque problème.

Au terme de ce chapitre, nous pouvons observer la différence d'impact entre la contrainte d'incompatibilité et l'introduction des tâches de traitement. Le graphe G_c change fondamentalement la structure des preuves et les différentes approches, et la manière d'ordonnancer les tâches d'acquisition. Alors qu'au contraire pour les tâches de traitement, les preuves sont similaires à celles déjà faites nécessitant juste un ajustement pour prendre en compte des tâches de traitement. Il va donc être intéressant d'observer l'évolution de la complexité des problèmes dans le Chapitre 6 lorsque nous introduisons les deux contraintes en même temps.

L'ensemble des résultats obtenus dans ce chapitre sont récapitulés dans le tableau 5.2.

Problème	Complexité	Référence
$\Pi_1^P : (a_i = b_i = p, L_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 69
$\Pi_2^P : (a_i = L_i = b_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 72
$\Pi_3^P : (a_i, L_i = L, b_i = b) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 72
$\Pi_4^P : (a_i = a, L_i = L, b_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 72
$\Pi_5^P : (a_i = a, L_i, b_i = b) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 73
$\Pi_6^P : (a_i = L_i = p, b_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	Polynomial	Page 73
$\Pi_7^P : (a_i, L_i = b_i = p) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	Polynomial	Page 80
$\Pi_8^P : (a_i = b_i = p, L_i = L) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	Polynomial	Page 80

Tab. 5.2 – Résumé des résultats du Chapitre 5

Chapitre 6

Prise en compte des tâches de traitement et du graphe de compatibilité

6.1 Introduction

Cette partie portera sur l'étude des problèmes d'ordonnancement avec tâches couplées en présence d'un **graphe de compatibilité quelconque** cette fois-ci et d'un **graphe de précédence** en ajoutant les tâches de traitement. Ces tâches ont toutes un temps d'exécution supérieur à 1. On garde le graphe de précédence particulier : chaque tâche d'acquisition est prédécesseur d'une tâche de traitement. De la même manière que dans les deux chapitres précédents, nous allons faire le même type d'étude sur la complexité mais cette fois-ci en présence des deux contraintes de précédence et d'incompatibilité. Nous pourrons ainsi voir que les techniques de réductions ou les algorithmes optimaux sont essentiellement basés sur le graphe de compatibilité, montrant ainsi l'impact de la contrainte d'incompatibilité. Enfin nous finirons avec une étude de l'approximation de certains problèmes critiques.

En reprenant la notation de la Figure 3.5 page 33, nous présentons sur la Figure 6.1 la vision globale de la complexité des mêmes problèmes d'ordonnancement avec la présence de tâches de traitement et d'un graphe de compatibilité. Les problèmes sont représentés à travers trois treillis comme précédemment où nous ajoutons la notation $\cup\mathcal{T}_i, G_c$ pour indiquer la présence respective des tâches de traitement et du graphe de compatibilité. Les problèmes étudiés dans ce chapitre seront notés Π_i^{PG}.

De la même manière que nous l'avons vu dans le Chapitre 4, la prise en compte de la contrainte d'incompatibilité rend les problèmes étudiés plus complexes qu'ils ne l'étaient. Ceci est toujours le cas lorsque les problèmes sont avec des tâches de traitement, ce qui entraîne les résultats de complexité suivant :

Remarque 6.1.1 :
Les problèmes qui étaient $\mathcal{NP}$-complets avec des tâches de traitement et un graphe de compatibilité complet restent $\mathcal{NP}$-complets avec des tâches de traitement et un ***graphe de structure quelconque (voir le Lemme 2.1.3 de la page 13)****.*

Nous retrouvons les mêmes changements de complexité que dans le Chapitre 4, les problèmes notés Π_1^{PG} et Π_3^{PG} sur la Figure 6.1 page 86 deviennent $\mathcal{NP}$-complets avec la prise en compte de la contrainte d'incompatibilité. Nous montrerons que la $\mathcal{NP}$-complétude de Π_1^{PG} entraîne celle de Π_3^{PG}. Le point négatif de l'introduction simultanée des deux contraintes est la difficulté dans

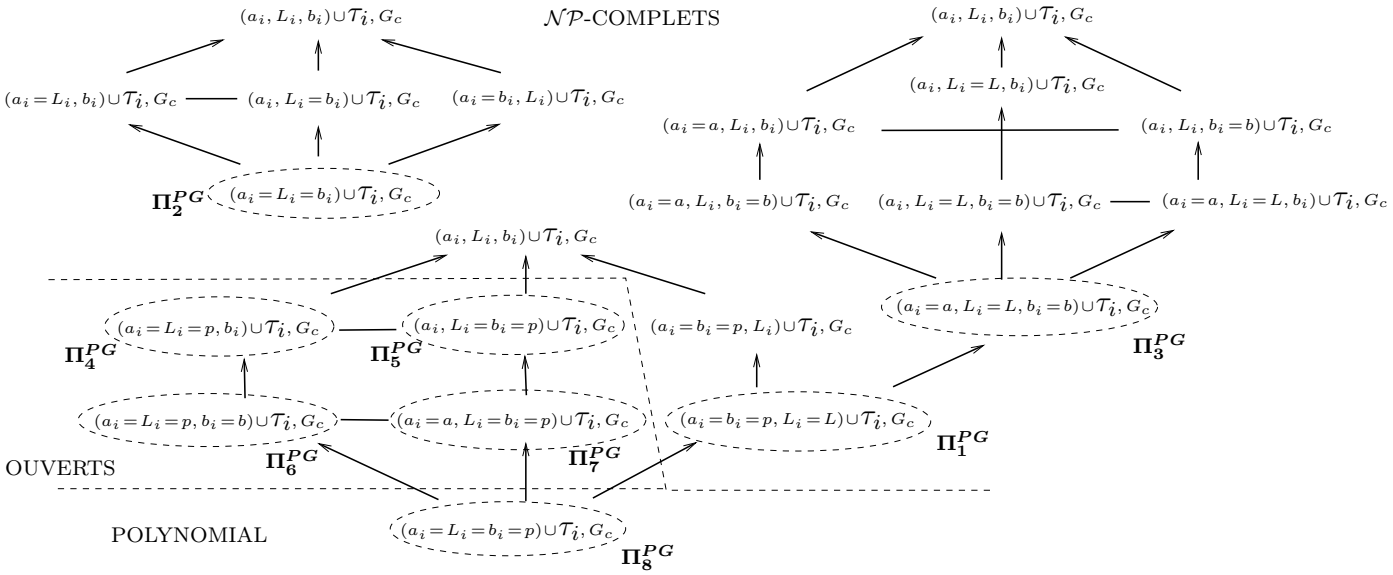


Fig. 6.1 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition et tâches de traitement en présence de contraintes de précédence et d'incompatibilité

l'analyse des problèmes Π_4^{PG}, Π_5^{PG}, Π_6^{PG} et Π_7^{PG} qui étaient polynomiaux. Ces problèmes ont été longuement étudiés, mais restent encore des problèmes ouverts.

Nous focaliserons notre étude sur les problèmes Π_i^{PG}, $i = 1, \ldots, 8$, qui se situent à la frontière de la $\mathcal{NP}$-complétude et la polynomialité selon le Schéma 6.1 (la complexité de Π_2^{PG} étant déjà donnée, nous nous intéresserons uniquement à son approximation). Pour ces problèmes là, nous fixerons certains paramètres afin de mieux évaluer l'influence du graphe de compatibilité et de l'introduction des tâches de traitement sur le passage de la polynomialité à la $\mathcal{NP}$-complétude.

6.1.1 Présentation du chapitre

Dans la première section nous donnerons les preuves de $\mathcal{NP}$-complétude et de polynomialité de quatre problèmes qui nous intéressent (pour certains nous étendrons l'analyse selon les valeurs des paramètres), puis nous proposerons les intuitions que nous avons par rapport aux quatre autres problèmes ouverts. Enfin dans la deuxième section, nous donnerons plusieurs algorithmes d'approximation avec garantie de performance pour les différents problèmes étudiés, en prenant en compte à nouveau des valeurs des paramètres selon les cas.

6.2 Classification de problèmes par la complexité

Nous allons montrer dans un premier temps la $\mathcal{NP}$-complétude de deux problèmes d'ordonnancement :

- $\Pi_1^{PG} : 1|prec, (a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.1 page 87)
- $\Pi_3^{PG} : 1|prec, (a_i = a, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.2 page 89)

Dans un deuxième temps nous nous intéresserons aux problèmes d'ordonnancement ouverts :

- $\Pi_4^{PG} : 1|prec, (a_i = L_i = p, b_i) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.3 page 89)
- $\Pi_5^{PG} : 1|prec, (a_i, L_i = b_i = p) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.4 page 90)
- $\Pi_6^{PG} : 1|prec, (a_i = L_i = p, b_i = b) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.5 page 90)

- Π_7^{PG} : $1|prec, (a_i = a, L_i = b_i = p) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.6 page 91)

Enfin, dans un troisième temps nous étudierons la polynomialité du problème d'ordonnancement Π_8^{PG} : $1|prec, (a_i = L_i = b_i = p) \cup (\tau_i, pmtn), G_c|C_{max}$ (Section 6.2.7 page 91)

6.2.1 Étude du problème $\Pi_1^{PG} : 1|(a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$, avec $L, p \in \mathbb{N}^*$

Nous avons montré dans le Chapitre 4 (resp. 5) que le problème $1|a_i = b_i = p, L_i = L, G_c|C_{max}$ (resp. $1|(a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c complet|C_{max}$) était $\mathcal{NP}$-complet. Intuitivement, l'association des deux contraintes devrait entraîner la même complexité.

Nous prenons un cas particulier de Π_1^{PG} dans lequel p est de longueur unitaire. Ce problème sera noté $\Pi_1^{GP'} : 1|(a_i = b_i = 1, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$. Si $\Pi_1^{PG'}$ est $\mathcal{NP}$-complet alors le cas général Π_1^{PG} sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3 de la page 13.

Nous allons aborder le problème $\Pi_1^{PG'}$ du point de vue de la complexité en faisant varier le paramètre L. Nous pouvons exhiber deux cas distincts, le premier est polynomial, et le second $\mathcal{NP}$-complet :

Lemme 6.2.1 :
Il existe un algorithme en temps polynomial permettant de résoudre le problème Π_1^{PG} avec $L = 1$.

Preuve :
Supposons que nous ayons n tâches d'acquisition et par définition du graphe de précédence n tâches de traitement. Le slot créé par chaque tâche d'acquisition ne permet pas de faire chevaucher plus de deux tâches en même temps, ainsi la recherche d'un couplage dans G_c nous permet d'ordonnancer les tâches saturées par le couplage deux à deux (créant ainsi des blocs avec aucun slot d'inactivité), pour deux tâches A_i et A_j nous avons $\sigma(A_j) = \sigma(A_i) + 1$. Après avoir ordonnancé les tâches correspondantes au couplage, nous exécutons le reste des tâches qui n'appartiennent pas au couplage de la même manière qu'au premier cas.

Cet ordonnancement engendre une solution optimale sans temps d'inactivité. En effet, pour un couplage maximum M de taille m, nous créons tout d'abord m blocs de taille 4 sans temps d'inactivité avec les tâches d'acquisition du couplage. Puis nous exécutons les $(n - 2m)$ tâches d'acquisition restantes créant ainsi chacune un slot d'inactivité. Les tâches de traitement étant par définition de taille supérieure à 1, tous les slots seront bouchés (Voir illustration sur la Figure 6.2).

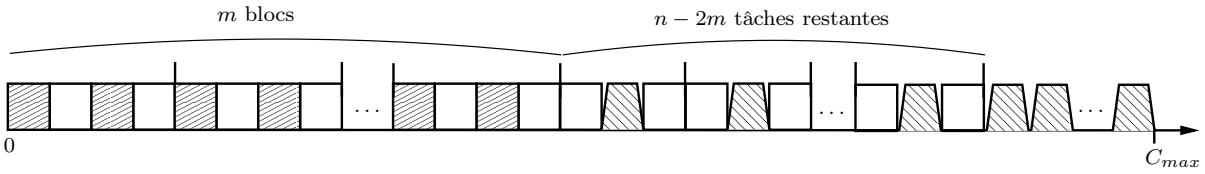


Fig. 6.2 – Illustration de l'algorithme donnant un ordonnancement sans temps d'inactivité

De manière plus simple, il suffit de prendre une arête dans G_c, d'exécuter en premier les tâches de l'arête puis toutes les autres. Les tâches de traitement rempliront tous les slots, et nous aurons bien un ordonnancement sans temps d'inactivité.

□

Lorsque $L \geq 2$, nous pouvons désormais faire chevaucher l'exécution de plus de deux tâches d'acquisition, ce qui nous conduit à rechercher des cliques dans le graphe de compatibilité pour limiter le temps d'inactivité sur le processeur. Nous allons montrer que la présence du graphe de compatibilité et de précédence entraîne la $\mathcal{NP}$-complétude.

Théorème 6.2.1 :
Le problème de décision $\Pi_1^{PG'}$ est $\mathcal{NP}$-complet.

Preuve :
Notre approche est similaire à la preuve proposée par Lenstra et al. [37] pour le problème $P|prec; p_j = 1|C_{max}$. Notre démonstration est basée sur le problème $\mathcal{NP}$-complet CLIQUE (Problème 2.2.2)

Soit I^* une instance du problème Clique, nous allons construire une instance I de Π avec $\boldsymbol{C_{max} = 2n + \sum_{T_i \in \mathcal{T}} \tau_i}$ de la manière suivante :

Soit $G = (V, E)$ un graphe, avec $|V| = n$:
- $\forall v \in V$, une tâche d'acquisition A_v est créée, composée de deux sous-tâches a_v et b_v avec leur temps d'exécution $a_v = b_v = 1$ et avec un temps d'inactivité entre ces deux sous-tâches, de longueur $L = (K - 1)$.
- Pour chaque arête $e = (v, w) \in E$, il y a une relation de compatibilité entre les deux tâches d'acquisition A_v et A_w.
- Pour chaque tâche A_v, nous créons une tâche de traitement T_v qui est son successeur.
- Chaque T_v a un temps d'exécution noté $\tau_v = L$.

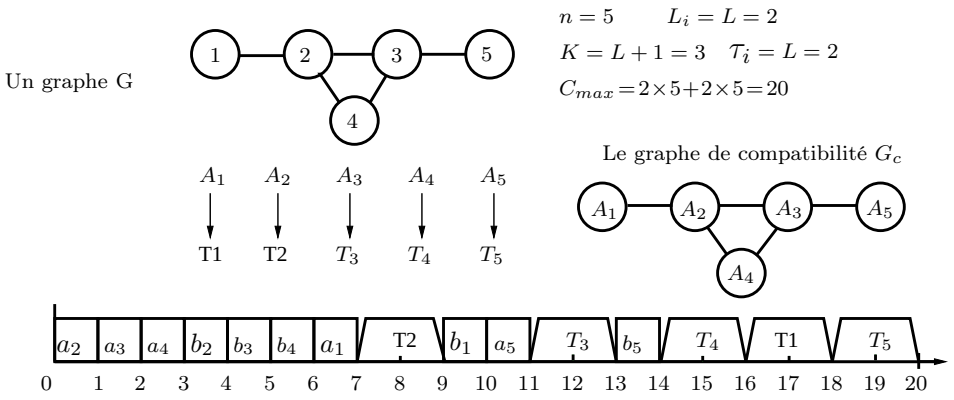


Fig. 6.3 – Illustration de la transformation en temps polynomial : CLIQUE $\propto \Pi$

- Supposons maintenant l'existence d'une clique de longueur $K = (L+1)$ dans le graphe G. Prouvons qu'il existe alors un ordonnancement sans temps d'inactivité. Nous exécutons les $K = (L+1)$ tâches d'acquisition de la clique à $t = 0$, formant ainsi un bloc sans temps d'inactivité. Nous exécutons ensuite les tâches d'acquisition restantes A_v, puis les tâches de traitement qui remplissent les temps d'inactivités des A_v.

- Réciproquement, supposons qu'il existe un ordonnancement sans temps d'inactivité, nous allons montrer que le graphe G contient une clique de taille $K = (L + 1)$. Avec les contraintes de précédence entre les tâches A_v et T_v, il est facile de voir que nous ne

pouvons ordonnancer qu'une tâche A_v à $t = 0$, $\forall v \in V$. Cette tâche va créer un temps d'inactivité de taille L, sachant que nous avons un ordonnancement sans temps d'inactivité, nous sommes sûr d'avoir $(L + 1)$ tâches d'acquisition exécutées les unes dans les autres.

Ainsi, nous avons $(L + 1)$ tâches d'acquisition qui sont compatibles. Et dans le graphe de compatibilité G_c, nous aurons une arête entre chaque couple de ces tâches A_v. En conséquence, les tâches $A_{p_1}, A_{p_2}, \ldots, A_{p_L}$, associées aux sommets du graphe G, forment une clique de taille $K = (L + 1)$.

Nous avons donc CLIQUE $\leq_T \Pi_1^{PG'}$, et nous savons que CLIQUE (Problème 2.2.2) est $\mathcal{NP}$-complet. Ainsi, d'après le Lemme 2.1.2 de la page 13, nous pouvons conclure que le problème $\Pi_1^{PG'}$ est $\mathcal{NP}$-complet.

□

L'étude de l'approximation du problème Π_1^{PG} se trouve à la Section 6.3.1 page 92, et une étude plus approfondie est donnée pour le cas $L = 2$ à la Section 6.3.2 page 95.

6.2.2 Étude du problème Π_3^{PG} : $1|(a_i = a, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c|C_{max}$

D'après les résultats obtenus dans le chapitre 5 (voir Figure 5.1), le problème $1|(a_i = a, b_i = b, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$ est toujours **ouvert**. En faisant varier les valeurs des paramètres fondamentaux a_i et b_i, nous pouvons proposer la remarque suivante :

Remarque 6.2.1 :
Le problème $\Pi_3^{PG} : 1|(a_i = a, b_i = b, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$ *est une généralisation du problème* $\Pi_1^{PG} : 1|(a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$. *En effet, en posant* $a = b = p$ *sur une instance de* Π_3^{PG} *nous obtenons une instance de* Π_1^{PG}.

Théorème 6.2.2 :
Le problème de décision associé Π_3^{PG} *est* $\mathcal{NP}$*-complet.*

Preuve :
La Remarque 6.2.1 implique la $\mathcal{NP}$-complétude du problème Π_3^{PG} d'après le Lemme 2.1.3 de la Section 2.1.4.

□

Nous venons de montrer que le problème $\Pi_3^{PG} : 1|(a_i = a, b_i = b, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$ était $\mathcal{NP}$-complet dans le cas général. Nous n'avons pas réussi à déterminer la classe de complexité lorsque $L < a + b$ ou $L \geq a + b$. Mais le fait que le problème soit polynomial pour $L < a + b$ lorsqu'il n'y a pas de tâches de traitement (voir Lemme 4.2.3 page 45) nous amène à poser la Conjecture 6.2.1.

Conjecture 6.2.1 *Le problème de décision associé* Π_3^{PG} *est polynomial lorsque* $L < a + b$.

L'étude de l'approximation du problème $\Pi_3^P G$ se trouve à la Section 6.3.3 page 103.

6.2.3 Étude du problème $\Pi_4^{PG} : (1|a_i = L_i = p, b_i) \cup (\tau_i, pmtn), G_c|C_{max}$, avec $p \in \mathbb{N}^*$

Nous n'avons toujours pas réussi à déterminer la classe de complexité de ce problème. Nous avons vu dans les chapitres précédents la polynomialité du problème en présence de la contrainte d'incompatibilité et sans tâches de traitement (Section 4.2.3 page 47), et également en présence

des tâches de traitement mais sans contrainte d'incompatibilité (Section 5.2.6 page 73). À la fin de cette deuxième étude, nous avons donné une remarque importante (Remarque 5.2.5 page 80) indiquant qu'un couplage maximum n'entraîne pas forcément une solution optimale. La plupart des problèmes de recherche de couplage maximal dans des graphes quelconques sont $\mathcal{NP}$-complets, ce qui nous amène à poser cette conjecture :

Conjecture 6.2.2 *Le problème de décision associé* Π_4^{PG} *est* $\mathcal{NP}$*-complet.*

6.2.4 Étude du problème $\Pi_5^{PG}:(1|a_i, L_i=b_i=p)\cup(\tau_i, pmtn), G_c|C_{max}$, avec $p \in \mathbb{N}^*$

Ce problème est composé de n tâches d'acquisition toutes sur le même modèle $(a_i, L_i=b_i=p)$. De la même manière que dans dans le chapitre avec la présence de la contrainte d'incompatibilité et sans tâches de traitement (Section 4.2.5 page 50), nous utilisons le **Théorème 4.2.6** donné par Orman et Potts pour obtenir la complexité du problème qui nous intéresse par symétrie à un autre problème. Le problème Π_5^{PG} est symétrique à Π_4^{PG}, mais comme nous n'avons pas su déterminer la classe de complexité de ce problème, nous ne pouvons que nous appuyer sur la Conjecture 6.2.2 pour en donner une autre.

Conjecture 6.2.3 *Le problème de décision associé* Π_5^{PG} *est* $\mathcal{NP}$*-complet sous la supposition que la Conjecture 6.2.2 soit vraie.*

6.2.5 Étude de $\Pi_6^{PG}:(1|a_i=L_i=p, b_i=b)\cup(\tau_i, pmtn), G_c|C_{max}$, avec $p \in \mathbb{N}^*$

La classification de Π_6^{PG} reste un problème ouvert. Nous avons vu dans les chapitres précédents la polynomialité du problème en présence de la contrainte d'incompatibilité et sans tâches de traitement (Section 4.2.3 page 47), et également en présence des tâches de traitement mais sans contrainte d'incompatibilité (Section 5.2.6 page 73). Ce problème est une spécification du problème ouvert Π_4^{PG}, où les secondes sous-tâches d'acquisition b_i sont toutes égales à une constante b. En contraignant ainsi les paramètres, nous simplifions le problème. Cette fois-ci l'intuition nous pousse à penser que la solution optimale du problème sera induit par un couplage de taille maximum suivant une fonction à minimiser ou maximiser.

Le point négatif, qui est ressorti de notre analyse, est la difficulté de trouver cette fonction. En effet, nous devons prendre en compte les slots créés par les tâches exécutées et les tâches de traitement disponibles, mais même pour un couplage de taille maximum donné, ordonnancer les couples en cherchant à mettre en premier ceux qui maximisent la somme des τ_i n'induit pas forcément un ordonnancement optimal comme le montre le contre-exemple 6.4.

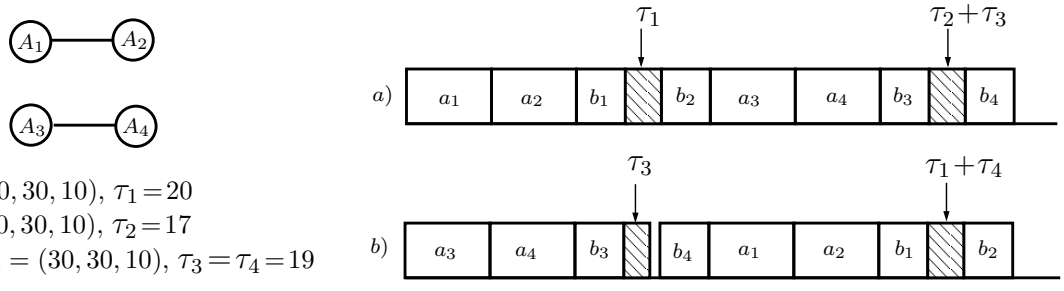


Fig. 6.4 – Contre-exemple pour le problème Π_6^{PG}. Chaque tâche A_i est équivalente au modèle $(a_i=L_i=p, b_i=b)$. Pour un couplage donné, l'ordonnancement $b)$ exécute les tâches de l'arête de poids maximum en premier mais crée un slot, alors que dans le cas $a)$ l'ordonnancement est sans temps d'inactivité si nous n'exécutons pas la tâche de poids maximum en première.

L'intuition de penser qu'un ordonnancement optimal peut être induit par un couplage de taille maximum minimisant ou maximisant une certaine fonction (qui devra prendre en compte la taille des tâches de traitement et les slots d'inactivité utilisés) nous amène à proposer cette conjecture :

Conjecture 6.2.4 *Le problème de décision* Π_6^{PG} *est polynomial.*

6.2.6 Étude de $\Pi_7^{PG} : (1|a_i = a, L_i = b_i = p) \cup (\tau_i, pmtn), G_c|C_{max}$, avec $p \in \mathbb{N}^*$

Ce problème est composé de n tâches d'acquisition toutes sur le même modèle ($a_i = a, L_i = b_i = p$). De la même manière que dans dans la Section 6.2.4, nous utilisons le **Théorème 4.2.6** donné par Orman et Potts [57] pour obtenir la complexité du problème qui nous intéresse par symétrie à un autre problème. Le problème Π_7^{PG} est symétrique à Π_6^{PG}, mais comme la classification de complexité de ce problème est ouvert, nous ne pouvons que nous appuyer sur la Conjecture 6.2.4 pour en donner une autre.

Conjecture 6.2.5 *Le problème de décision* Π_7^{PG} *est polynomial sous la supposition que la Conjecture 6.2.4 soit vrai.*

6.2.7 Étude de $\Pi_8^{PG} : (1|a_i = L_i = b_i = p) \cup (\tau_i, pmtn), G_c|C_{max}$, avec $p \in \mathbb{N}^*$

Ce problème est le cas le plus simple des problèmes que nous pouvons rencontrer, toutes les tâches sont équivalentes, et les paramètres fondamentaux sont tous égaux à la même constante $p \in \mathbb{N}^*$. Nous proposons la classification suivante pour ce problème :

Théorème 6.2.3 :
Le problème de décision Π_8^{PG} *est polynomial.*

Preuve :
La preuve est triviale, il suffit de chercher un couplage de taille maximum de poids maximum où chaque arête est pondérée par la somme des tâches de traitement associées aux sommets de l'arête. Dans le cas où il existe un couplage parfait, il est inutile de pondérer les arêtes car l'ordonnancement est sans temps d'inactivité.

□

6.2.8 Vision globale de la complexité

Nous avons montré la $\mathcal{NP}$-complétude des deux problèmes Π_1^{PG} et Π_3^{PG}, et la polynomialité de Π_8^{PG}. Comme nous l'avons indiqué dans l'introduction de ce chapitre, tous les problèmes qui étaient déjà $\mathcal{NP}$-complets avec des tâches de traitement et un graphe de compatibilité complet (voir Figure 5.1) restent $\mathcal{NP}$-complets lorsque G_c a une structure quelconque (voir Figure 6.1).

La classification des problèmes Π_i^{PG}, $i = 4, \ldots, 7$, qui étaient polynomiaux avec un graphe de compatibilité complet, reste ouverte. Nous proposons toutefois des conjectures sur le fait que les problèmes Π_4^{PG} et Π_5^{PG} seraient $\mathcal{NP}$-complets, et Π_6^{PG} et Π_7^{PG} seraient polynomiaux.

Ceci finit la classification des problèmes d'ordonnancement avec tâches-couplées en présence d'un graphe de compatibilité ayant une structure quelconque. Dans la section suivante, nous allons continuer l'analyse des problèmes Π_1^{PG}, Π_2^{PG} et Π_3^{PG}, en étudiant différentes heuristiques d'approximation avec garantie de performance.

6.3 Étude de l'approximation

Cette section portera sur l'étude d'heuristiques d'approximation focalisées sur les problèmes $\mathcal{NP}$-complets Π_1^{PG}, Π_2^{PG} et Π_3^{PG} lorsque les tâches de traitement ont toutes un temps d'exécution égal à 1 (ce qui revient au pire des cas). Puis, nous étudierons en détail les algorithmes d'approximation pour des cas très proche de la polynomialité selon les paramètres fondamentaux, cherchant ainsi la limite entre la polynomialité et la $\mathcal{NP}$-complétude.

6.3.1 Algorithme d'approximation de Π_1^{PG}

Intéressons nous à l'approximation du problème $\mathcal{NP}$-complet Π_1^{PG}, l'étude de la complexité de ce problème a été faite à la Section 6.2.1 page 87. Nous allons développer un algorithme d'approximation de complexité polynomiale avec garantie de performance non triviale. Nous étudierons la garantie de performance classique c'est à dire le ratio, pris sur toutes les instances, entre la solution proposée par une heuristique et une solution optimale. Cet algorithme est basé sur le problème de la recherche d'un couplage maximum dans le graphe de compatiblité.

Remarque 6.3.1 *Notons que dans le cas où le temps d'exécution des tâches de traitement est supérieur à un ($\tau_i > 1,\ \forall i$), la somme des temps d'inactivité ne peut pas être plus élevée que lorsque les délais d'exécution des tâches de traitement est égal à 1 ($\tau_i = 1$).*

Par la suite, les tâches de traitement auront un temps d'exécution égal à $\tau_i = 1$, pour $i = 1, \ldots, n$.

Borne inférieure pour toute solution optimale

Lemme 6.3.1 *Une borne inférieure basée sur la taille d'un couplage maximum pour Π_1^{PG} est égale à $C_{max}^{opt} \geq max\{3n, (n-2m)(L+2)+1\}$*

Preuve :
Nous allons proposer deux bornes inférieures. Pour la première, l'ordonnancement optimal est obtenu lorsque nous n'avons aucun temps d'inactivité. Par ailleurs, nous savons que le nombre de tâches de traitement est égal au nombre de tâches d'acquisition et que dans le pire des cas, toutes les tâches de traitement ont un temps d'exécution égal à $\tau_i = 1$, $\forall i$. Ainsi, nous avons : $C_{max}^{opt} \geq T_{seq} = 2n + \sum_{\tau_i \in \mathcal{T}} \tau_i = 2n + n = 3n$.

De plus, en considérant un couplage maximum M dans le graphe de compatibilité de taille m, le nombre de sommets indépendants s'élèvent à $(n-2m)$.

Dans le pire des cas, l'ordonnancement optimal est forcément supérieur à l'ordonnancement des sommets indépendants suivis de la dernière tâche de traitement. Et donc nous avons :
$C_{max}^{opt} \geq (n-2m)(L+2)+1$.

Pour notre étude, la borne inférieure sera $\boldsymbol{C_{max}^{opt} \geq max\{3n, (n-2m)(L+2)+1\}}$.

□

Borne supérieure de l'algorithme

Algorithme 8 : Un algorithme d'approximation en temps polynomial

Données : $\mathcal{A}$, $\mathcal{T}$, G_c, $L \geq 1$
Résultat : C^h_{max}
début
- Chercher un couplage de taille maximum M dans G_c
- Pour chaque arête (i, j) du couplage de taille maximum, les tâches d'acquisition A_i et A_j sont ordonnancées telles que $\sigma(A_j) = \sigma(A_i) + 1$
- Pour chaque sommet i restant, exécuter la tâche d'acquisition A_i
- Exécuter les tâches de traitement aux premiers emplacements libres en respectant les contraintes de précédence

fin

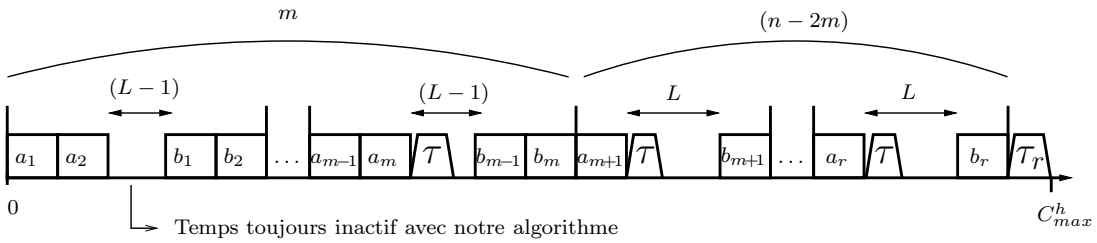


Fig. 6.5 – Illustration de l'algorithme d'approximation

La complexité de cet algorithme dépend surtout de celle du couplage de taille maximum. En effet, après avoir trouvé le couplage maximum, les autres étapes de l'algorithme se font en temps linéaire. En prenant l'algorithme donné par Gabow [30], nous avons une complexité de $O(n^3)$ pour le couplage maximum. Ainsi l'**algorithme** 8 s'exécutera en $\boldsymbol{O(n^3)}$.

Lemme 6.3.2 *L'algorithme 8 donne une borne supérieure pour* Π_1^{PG} *égale à* $C^h_{max} \leq T_{\text{seq}} + T_{\text{idle}} \leq [2n + max\{n - nL + m(L+1) + L - 1, 1 + \mathbb{1}_{\{m=\frac{n}{2}\}}\}] + [L(n-m) - m]$ [16]

Preuve :
Nous allons donner quelques remarques essentielles sur la structure de l'ordonnancement donné par notre algorithme d'approximation. Nous supposons que nous avons un ordonnancement donné par l'algorithme d'approximation avec un couplage maximum de taille m.

- Dans le premier bloc créé par deux tâches d'acquisition reliées dans le couplage maximum, il existe un temps d'inactivité de taille $(L-1)$.
- Pour deux tâches M-saturées exécutées consécutivement, le temps d'inactivité de ces tâches est $(L-1)$.
- Pour chaque tâche d'acquisition M-non-saturée restante dans le couplage maximum, le temps d'inactivité est de L.
- Considérons la dernière tâche d'acquisition notée A_r. Après son exécution, nous pouvons ordonnancer la tâche de traitement τ_r (ce cas survient lorsque toutes les tâches de traitement, à l'exception de τ_r, sont ordonnancées avant le temps de complétude de A_r. Ajoutons que nous pouvons avoir deux tâches de traitement exécutées lorsque nous avons un couplage parfait), ou plusieurs tâches de traitement (ce cas se produit lorsqu'il n'y a pas de temps d'inactivité avant le temps de complétude de A_r). Voir figure (6.5), pour

[16] $\mathbb{1}_{\{m=\frac{n}{2}\}}$ est la fonction indicatrice qui indique si le couplage est parfait.

une illustration du cas où la tâche τ_r est la seule tâche de traitement exécutée après le temps de complétude de A_r). Ainsi, le nombre de tâches de traitement exécutées après A_r est $max\{n-(m-1)(L-1)-(n-2m)L, 1+\mathbb{1}_{\{m=\frac{n}{2}\}}\}$.

Finalement, notre borne supérieure sera :
$C^h_{max} \leq T_{\text{seq}} + T_{\text{idle}} \leq [2n + max\{n - nL + m(L+1) + L - 1, 1 + \mathbb{1}_{\{m=\frac{n}{2}\}}\}] + [L(n-m) - m]$
□

Performance relative

Grâce à une étude du max, nous pouvons étudier le comportement de la performance relative du problème Π_1^{PG} que nous obtenons en utilisant cet algorithme.

Lemme 6.3.3 *L'algorithme 8 donne une performance relative pour* Π_1^{PG} *égale à* $\rho \leq \frac{(L+6)}{6} - \frac{1}{2(L+2)} + \frac{(L+3)}{6n(L+2)}$.

Preuve :
En premier lieu, nous constatons que pour $L \geq 4$, il est facile de voir que $max\{n - nL + m(L+1) + L - 1, 1 + \mathbb{1}_{\{m=\frac{n}{2}\}}\} = 1 + \mathbb{1}_{\{m=\frac{n}{2}\}}$ puisque par définition $m \leq \frac{n}{2}$. Ainsi $C^h_{max} \leq 2n + L(n-m) - m + 1 + \mathbb{1}_{\{m=\frac{n}{2}\}}$. Du coup, dans un première temps, le tableau suivant donne un résumé du rapport de la performance relative ρ pour $L \in \{1,2,3\}$ où la valeur de C^h_{max} varie selon L.

	$L=1$	$L=2$	$L=3$
$\boldsymbol{\rho}$	1	si $m \geq \frac{n}{3}$, alors $\boldsymbol{\rho \leq 1 + \frac{1}{3n}}$ si $\frac{n+1}{8} \leq m < \frac{n}{3}$, alors $\boldsymbol{\rho \leq \frac{4}{3}}$ si $m < \frac{n+1}{8}$, alors $\boldsymbol{\rho \leq \frac{29}{24} + \frac{5}{24n}}$	si $m = \frac{n}{2}$, alors $\boldsymbol{\rho \leq 1 + \frac{2}{3n}}$ si $\frac{2n+1}{10} \leq m < \frac{n}{2}$, alors $\boldsymbol{\rho \leq \frac{7}{5} + \frac{1}{5n}}$ si $m < \frac{2n+1}{10}$, alors $\boldsymbol{\rho \leq \frac{7}{5} + \frac{1}{5n}}$

TAB. 6.1 – Performance relative pour $\alpha \in \{1,2,3\}$

Dans un deuxième temps, nous pouvons nous focaliser sur l'étude de $L \geq 4$ où $C^h_{max} \leq 2n + L(n-m) - m + 1 + \mathbb{1}_{\{m=\frac{n}{2}\}}$. De plus, puisque $C^{opt}_{max} \geq max\{3n, nL + 2n - 2mL - 4m + 1\}$, les cas suivants doivent être considérés :

- Pour $m \in [0, \frac{n(L-1)+1}{2(L+2)}[$, $C^{opt}_{max} \geq nL + 2n - 2mL - 4m + 1$.
- Pour $m \in [\frac{n(L-1)+1}{2(L+2)}, \frac{n}{2}]$, $C^{opt}_{max} \geq 3n$.

Selon les valeurs de m, nous donnons la borne supérieure pour l'ordonnancement proposé par l'heuristique h, et la borne inférieure pour un ordonnancement optimal (voir illustration Figure 6.6).

Notons que pour $m = 0$, $\rho = 1$ (c'est évident, puisque le graphe de compatibilité est un ensemble de tâches indépendantes), de plus pour $m = \frac{n}{2}, \rho = \frac{(L+3)}{6} + \frac{1}{3n}$.

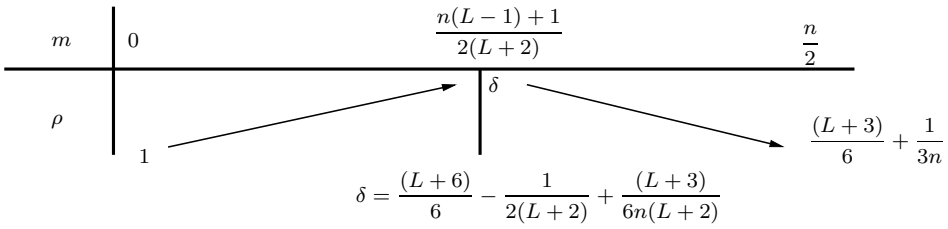


Fig. 6.6 – Comportement de la performance relative ρ en fonction de m

Puisque le pire cas se produit lorsque $\boldsymbol{m = \frac{n(L-1)+1}{2(L+2)}}$, nous en déduisons que la performance relative ρ est bornée par $\boldsymbol{\frac{(L+6)}{6} - \frac{1}{2(L+2)} + \frac{(L+3)}{6n(L+2)}}$.

□

Du fait que le problème CLIQUE (ou CLIQUE COVER) soit difficile à approximer, et que tout calcul de bornes supérieures selon différentes heuristiques porte sur le graphe G_c et essentiellement sur la variable L, nous proposons la conjecture suivante :

Conjecture 6.3.1 *Il n'est pas possible de trouver un algorithme permettant d'obtenir une garantie de performance bornée par une constante.*

6.3.2 Étude d'approximation de Π_1^{PG} lorsque $L = 2$

Nous allons nous focaliser sur l'approximation du problème Π_1^{PG} lorsque le temps d'inactivité de chaque tâche d'acquisition est égal à 2. Nous allons présenter dans un premier temps la particularité de la structure des tâches d'acquisition, puis nous proposerons des heuristiques avec garantie de performance non triviale pour ce problème.

Discussion sur l'ordonnancement possible des tâches d'acquisition

L'étude du problème dépend de deux points essentiels, la structure des tâches d'acquisition et le graphe de compatibilité G_c. Les contraintes particulières de ce type de structure amènent des problèmes de recouvrement de sommets dans le graphe G_c. Nous ne pouvons pas faire chevaucher plus de trois tâches d'acquisition à la fois, entraînant ainsi seulement quatre type d'ordonnancements possibles pour les tâches d'acquisition (voir Figure 6.7).

Observation 6.3.1 :
Le temps d'inactivité entre les deux sous-tâches restreint les possibilités d'ordonnancement. En effet sur la Figure 6.7, nous pouvons voir les quatre types d'ordonnancement, et par équivalence les quatre types de recouvrement des sommets de G_c. Pour chaque cas, nous avons au plus deux slots et plus de deux tâches de traitement exécutables après les tâches d'acquisition. Donc, si nous exécutons les triangles, les chaînes, et les arêtes les unes après les autres, il n'y aura aucun temps d'inactivité (excepté pour le premier slot s'il n'y a pas de triangle). Les seuls slots d'inactivité que nous pouvons avoir viennent des sommets isolées C_0. Ces tâches seront les dernières à être exécutées en raison de leur structure.

Ces quatre façons d'ordonnancer impliquent donc quatre types de recouvrement des sommets dans G_c :
- Les sommets non recouvert notés C_0
- Les arêtes notées C_1

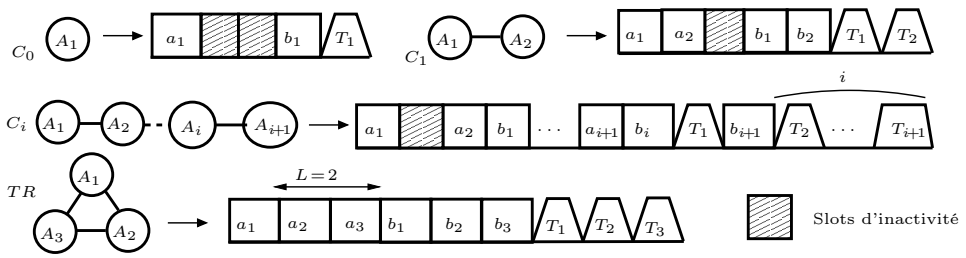


Fig. 6.7 – Illustration des quatre types d'ordonnancement.

– Les chaînes de longueur $k > 1$ notées C_k
– Les triangles notés TR

La présence des triangles dans G_c va augmenter la difficulté de notre étude. Pour de meilleurs résultats d'approximation, le problème Π_1^{PG} avec $L = 2$ est séparé en deux cas, dépendant du fait que G_c contienne des triangles ou non. Nous noterons les deux cas $\boldsymbol{\Pi_1^{PG} - TR}$ (lorsqu'il n'y aura pas de triangle) et $\boldsymbol{\Pi_1^{PG} + TR}$ (lorsqu'il y en aura).

Le problème $\Pi_1^{PG} - TR$ reste $\mathcal{NP}$-complet lorsqu'il n'y a pas de triangle. En effet résoudre ce problème peut revenir sur certaine instance à chercher une chaîne hamiltonienne dans G_c.

Avec les hypothèses précédentes, il est clair qu'une bonne heuristique pour ces deux problèmes sera basée sur le recouvrement de sommets de G_c, mais d'après Steiner [67] ou Masuyama [52] nous savons que recouvrir les sommets d'un graphe par des chaînes de longueurs différentes plus grandes que deux est un problème $\mathcal{NP}$-complet. Dans le but d'obtenir dans les deux cas un bon algorithme d'approximation en temps polynomial, nous allons utiliser la même approche basée sur un **2-recouvrement** défini dans la Section 2.3 à la page 16.

Borne inférieure pour $\boldsymbol{\Pi_1^{PG} - TR}$

Nous allons nous intéresser au cacul d'une borne inférieure la plus exacte possible. Dans le cas sans triangle, il y aura toujours un slot d'inactivité lorsque nous ordonnancerons en premier les tâches d'acquisition couvertes dans G_c par une arête ou une chaîne. D'après l'observation 6.3.1, nous allons calculer le nombre de slots d'inactivité obtenu après l'exécution des n tâches d'acquisition et de traitement. Dans ce but, prenons un recouvrement noté Sol_1 pour un ordonnancement optimal. Dans ce recouvrement, nous proposons les définitions suivantes :

– Soit $\boldsymbol{Nb(C_i)}$ le nombre de chaîne C_i.
– Soit $\boldsymbol{n_2}$ le nombre de sommets couverts dans la solution optimale (voir Figure 6.8).
– Le nombre de sommets est égal à $\boldsymbol{n = Nb(C_0) + n_2}$
– Soit $\boldsymbol{f}$ la fonction dépendant des variables $Nb(C_i)$ et comptant le nombre de slots d'inactivité (excepté pour le premier représenté par un **-1** dans l'équation) dans l'ordonnancement, après l'exécution des tâches de traitement dans les slots créés par les tâches d'acquisition :

$f =$ slots d'inactivité des sommets non couverts $-$ les tâches de traitement restantes après l'exécution des chaînes

$$= 2Nb(C_0) + Nb(C_1) + (2\sum_{i=2}^{n_2-1} Nb(C_i) - \mathbf{1}) - (Nb(C_0) - 1) - 2Nb(C_1) - \sum_{i=2}^{n_2-1} [(i+1)Nb(C_i)]$$

$$= Nb(C_0) - Nb(C_1) - \sum_{i=2}^{n_2-1} [(i-1)Nb(C_i)]$$

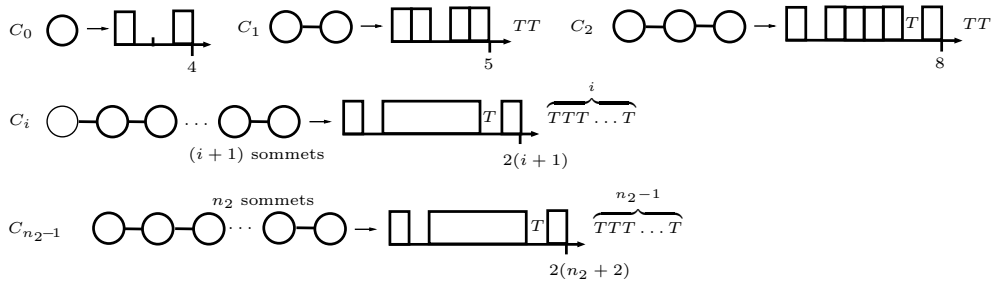


Fig. 6.8 – Différentes chaînes de la couverture de G_c

Selon les valeurs de la fonction f, la borne inférieure peut être déduite :

Lemme 6.3.4 *La borne inférieure est égale à* $C^{opt}_{max} \geq T_{sequentiel} + T_{idle} = 3n + 1 + max\{0, f\}$.

Intéressons nous à la fonction f, qui calcule le nombre de slots d'inactivité restants après exécution des tâches de traitement. Nous en déduisons le Lemme 6.3.5.

Lemme 6.3.5 :
Il existe une solution optimale de $\Pi_1^{PG} - TR$ *minimise* $Nb(C_0)$.

Preuve :
Il est évident de voir que le fait de minimiser f fournit une solution optimale à $\Pi_1^{PG} - TR$. Montrons que si $Nb(C_0)$ est minimisé dans f, la valeur $Nb(C_1) + \sum_{i=2}^{n_2-1} (i-1)Nb(C_i)$ sera augmentée. Supposons donc que nous avons $Nb(C_0)$ sommets non recouverts dans un recouvrement M non maximum[17].

Considérons alors que M couvre un sommet de plus noté x_0, il y a deux possibilités :

1. Si x_0 est connecté dans G_c à une arête de M, une chaîne est créée dans M et f diminue. En effet, l'exécution d'une arête plus celle d'un sommet isolé crée en tout trois slots d'inactivité, tandis qu'une chaîne en crée seulement deux.
2. Si x_0 est connecté dans G_c à un noeud d'une chaîne de M, deux chaînes sont créées dans M et f diminue. Il est clair que le fait de couper une chaîne dans le but de créer deux chaînes (arêtes inclues) entraîne la création de quatre slots.

Donc minimiser le nombre de sommets isolés entraîne que la fonction f diminue ou reste à la même valeur. Donc minimiser $Nb(C_0)$ revient à obtenir un ordonnancement optimal.

□

Remarque 6.3.2 :
D'après le Lemme 6.3.5, le recouvrement Sol_1*, associé à un ordonnancement optimal, couvre un maximum de sommets. Dans la suite, nous dirons que cette couverture est maximum.*

[17] Un recouvrement maximum couvre un maximum de sommets avec un minimum de chaînes.

Borne supérieure pour $\Pi_1^{PG} - TR$

Nous allons maintenant nous intéresser à la borne supérieure obtenue lorsque nous utilisons une heuristique basée sur la recherche d'un 2-recouvrement. Les lemmes suivants permettent d'obtenir un lien entre notre recouvrement et celui d'une solution optimale.

Lemme 6.3.6 :
Un 2-recouvrement minimise le même nombre de sommets isolés dans G_C que n'importe quelle couverture maximum composée de chaînes de longueur différentes (chaîne de longueur 1 acceptées).

Preuve :
La preuve est triviale, en effet n'importe quelle chaîne peut être découpée en arêtes et en chaînes de longueur 2.

□

D'après le Lemme 6.3.6, nous savons que le nombre de sommets isolés par un 2-recouvrement est également égale au nombre $Nb(C_0)$ de sommets isolés dans une solution optimale. Donc l'objectif est de couper les chaînes ayant une longueur plus grande que 2 en arêtes et en chaînes de longueur 2. Nous obtenons alors deux nouvelles quantités :

1. Soit $\boldsymbol{Nb^h(C_1)}$ le nombre d'arêtes dans notre heuristique.
2. Soit $\boldsymbol{Nb^h(C_2)}$ le nombre de chaînes de longueur 2 dans notre heuristique.

Afin d'affiner notre borne supérieure, nous allons montrer que pour un 2-recouvrement donné, le fait d'exécuter une arête, qui crée un slot d'inactivité et donne deux tâches de traitement, est plus intéressant que d'exécuter une chaîne de longueur 2 qui crée un slot d'inactivité non rempli et donne deux tâches de traitement (car la troisième est utilisée pour remplir un slot).

Lemme 6.3.7 :
Un 2-recouvrement maximum consistant à minimiser en premier $Nb(C_0)$, puis maximiser $Nb(C_1)$, fournit un meilleur ordonnancement.

Preuve :

Avec seulement des arêtes et des chaînes des longueur 2, nous avons la fonction $f = Nb(C_0) - Nb(C_1) - Nb(C_2) = Nb(C_0) - \left[\frac{n_1}{2} + \frac{n_2}{3}\right]$, où n_1 (resp. n_2) est le nombre de sommets couverts par des arêtes (resp. par des chaînes de longueur 2). Un 2-recouvrement maximum minimise $Nb(C_0)$, ainsi dans le but d'obtenir un meilleur ordonnancement, nous devons minimiser la fonction f, et le mieux reste de maximiser $\left[\frac{n_1}{2} + \frac{n_2}{3}\right]$. Donc, nous devons maximiser le nombre d'arêtes n_1.
□

Nous en déduisons la valeur de la borne supérieure dans le pire des cas :

Lemme 6.3.8 *La borne supérieure est égale à* $C^h_{max} \leq 3n + 1 + max\{0, Nb(C_0) - Nb^h(C_1) - Nb^h(C_2)\}$

Preuve :
À partir de la solution optimale Sol_1, pour chaque chaîne C_i de longueur impaire (resp. paire),

nous obtenons ($\frac{i+1}{2}$) arêtes (resp. ($\frac{i-2}{2}$) arêtes et une chaîne de longueur 2) par découpage. Ce qui donne :

$$\begin{aligned} Nb^h(C_1) &\leq Nb(C_1) + \sum_{i(odd)=3}^{n_2-1} [(\frac{i+1}{2})Nb(C_i)] + \sum_{i(even)=2}^{n_2-1} [(\frac{i-2}{2})Nb(C_i)] \\ &= Nb(C_1) + \sum_{i=2}^{n_2-1} [(\frac{i+1}{2})Nb(C_i)] - \sum_{i(even)=2}^{n_2-1} \frac{3}{2} Nb(C_i) \qquad (6.1) \\ Nb^h(C_2) &\leq \sum_{i(even)=2}^{n_2-1} Nb(C_i) \qquad (6.2) \end{aligned}$$

Donc, la longueur du makespan est égale au temps séquentiel plus le temps d'inactivité non rempli par les tâches de traitement. La borne supérieure C^h_{max} est donc égale à $C^h_{max} \leq 3n+1+max\{0, Nb(C_0)-Nb^h(C_1)-Nb^h(C_2)\}$.

□

Performance relative pour $\Pi_1^{PG} - TR$

Théorème 6.3.1 :
Il existe un algorithme en temps polynomial avec une performance relative bornée par $\frac{13}{12}$ *pour le problème* $\Pi_1^{PG} - TR$.

Preuve :
Nous allons étudier la performance relative ρ en fonction de la variable $Nb(C_0)$. Tout d'abord, notons que $n = Nb(C_0)+2Nb(C_1)+\sum_{i=2}^{n_2-1}(i+1)Nb(C_i)$, et d'après l'équation 6.3.4 le pire cas arrive lorsque $Nb(C_0) = Nb(C_1)+\sum_{i=2}^{n_2-1}[(i-1)Nb(C_i)]$ (voir Figure 6.9).

Avec les substitutions de $Nb(C_0)$ et n dans les bornes C^{opt}_{max} et C^h_{max}, nous obtenons :

• $C^{opt}_{max} \geq 3n+1 = 1+3Nb(C_0)+6Nb(C_1)+3\sum_{i=2}^{n_2-1}[(i+1)Nb(C_i)]$

$= 1 + 9Nb(C_1) + 6\sum_{i=2}^{n_2-1}[iNb(C_i)]$

• $C^h_{max} \leq 3n+1+ Nb(C_0)-Nb^h(C_1)-Nb^h(C_2)$

$= 3n+1+\sum_{i=2}^{n_2-1}\left[\frac{(i-3)}{2}Nb(C_i)\right]+\frac{1}{2}\sum_{i(pair)=2}^{n_2-1} Nb(C_i) \leq 3n+1+\frac{1}{2}\sum_{i=2}^{n_2-1}[iNb(C_i)]$

Ainsi le ratio de performance relative est le suivant :

$$\rho \leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{3n+1+\frac{1}{2}\sum_{i=2}^{n_2-1}[iNb(C_i)]}{3n+1} = 1+\frac{\frac{1}{2}\sum_{i=2}^{n_2-1}[iNb(C_i)]}{1+9Nb(C_1)+6\sum_{i=2}^{n_2-1}[iNb(C_i)]} \leq \mathbf{\frac{13}{12}}$$

□

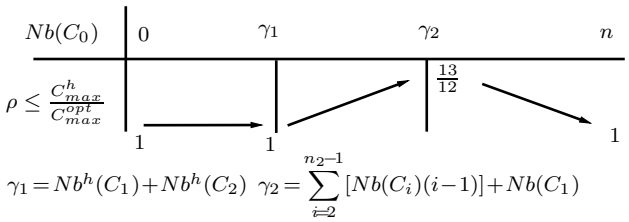


Fig. 6.9 – Variation de $\boldsymbol{\rho}$ selon les valeurs de $Nb(C_0)$

L'algorithme 9 consiste à minimiser en premier la variable $Nb(C_0)$, puis maximiser $Nb(C_2)$ à partir d'un 2-recouvrement. Ce qui permet d'obtenir une $\frac{\mathbf{13}}{\mathbf{12}}$-approximation pour $\Pi_1^{PG} - TR$.

Algorithme 9 : Utilisation d'un 2-recouvrement pour $\Pi_1^{PG} - TR$

Données : $G_c = (V, E)$, avec $|V| = n$
Résultat : C^h_{max} longueur de l'ordonnancement obtenue avec notre heuristique
début
  M_1 := Couplage de taille maximum dans G_c
  M_2 := 2-recouvrement de taille maximum à partir de M_1
  $M_3 := Transformation(M_2)$
  Ordonnancer en premier les sommets couverts par les arêtes de M_3, puis les chaînes de M_3
  Ordonnancer les sommets isolés à la suite, et finir en exécutant les tâches de traitement au premier slot d'inactivité disponible selon les tâches
fin

Remarque 6.3.3 *$Transformation(M_2)$ est l'opération réalisée sur M_2 qui transforme en temps polynomial les chaînes de longueur 2 en arêtes, lorsque cela est possible. L'opération est la suivante :*

1. *Chaque chaîne de longueur 2 dans M_2 est contractée en un seul sommet, tout en gardant les arêtes dans G_c qui relient les extrémités de la chaîne.*
2. *Nous cherchons un couplage maximum dans ce nouveau graphe avec les sommets contractés.*
3. *Les sommets contractés redeviennent des arêtes (illustration Figure 6.10).*

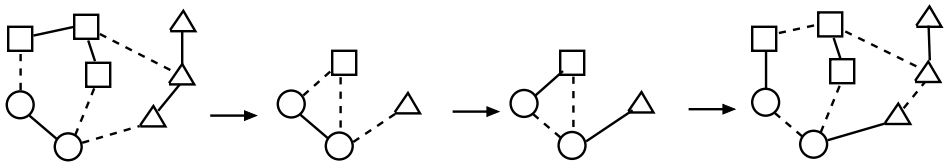


Fig. 6.10 – Illustration de la transformation des chaînes de longueur 2 en arêtes

Borne inférieure pour $\Pi_1^{PG} + TR$

Intéressons nous maintenant au calcul de la borne inférieure dans le cas avec triangles. Cette étude est similaire à la précédente. Dans le cas présent, nous devons simplement ajouter les triangles à l'ensemble de toutes les chaînes dans la couverture de G_c, dans le but d'obtenir un

ordonnancement optimal. Soit $\boldsymbol{Nb(TR)}$ le nombre de triangles dans le recouvrement, dénoté Sol_3 cette fois-ci, associé à un ordonnancement optimal. L'ajout des triangles à la solution optimale entraîne plusieurs changements :

- $n = Nb(C_0) + 2Nb(C_1) + \sum_{i=2}^{n_2-1}(i+1)Nb(C_i) + 3Nb(TR)$
- La borne inférieure devient[18] :

$$C^{opt}_{max} \geq 3n + \mathbb{1}_{\{Nb(TR)=0\}} + max\{0, Nb(C_0) - Nb(C_1) - \sum_{i=2}^{n_2-1}[Nb(C_i)(i-1)] - 3Nb(TR)\} \quad (6.3)$$

Contrairement au cas précédent sans triangle, dans le cas présent le fait de minimiser $Nb(C_0)$ n'implique pas la minimisation de $f = Nb(C_0) - Nb(C_1) - \sum_{i=2}^{n_2-1}[Nb(C_i)(i-1)] - 3Nb(TR)$. En effet, pour certaines instances particulières, il est préférable de laisser un sommet non couvert dans le but d'avoir un triangle et ainsi obtenir un ordonnancement sans temps d'inactivité (voir Figure 6.11). Donc le nombre $Nb^h(C_0)$ de sommets isolés par le recouvrement de notre heuristique est inférieur ou égal au nombre $Nb(C_0)$ de sommets isolés de la solution optimale. Afin de simplifier les calculs de bornes ou de performance relative, nous nous plaçons dans le pire des cas lorsque $Nb^h(C_0) = Nb(C_0)$.

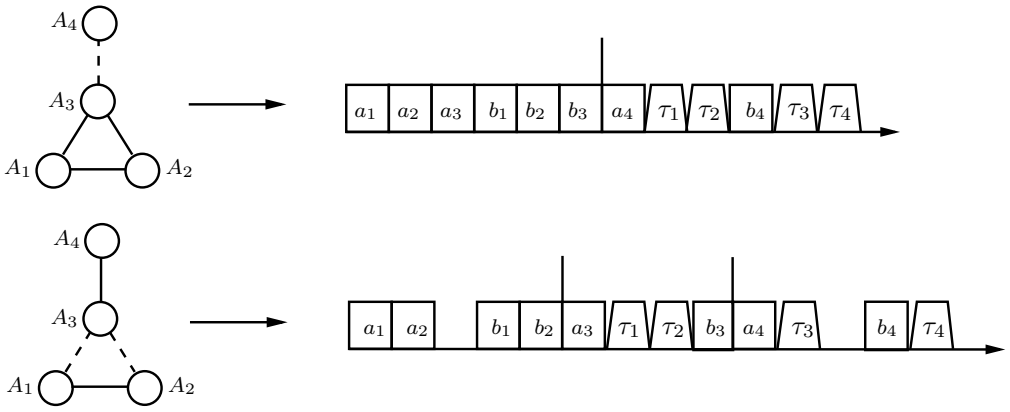


Fig. 6.11 – Illustration du fait que minimiser les sommets isolés ne donne pas toujours l'optimal

Borne supérieure pour $\Pi_1^{PG} + TR$

En utilisant toujours le même algorithme d'approximation basé sur un 2-recouvrement, nous allons calculer la borne supérieure obtenue. De la même manière que précédemment, nous cherchons à calculer au pire cas le nombre d'arêtes et de chaînes de longueur 2 obtenu par notre heuristique en fonction du nombre de chaînes et d'arêtes dans la solution optimale. Toutefois, la présence de triangles dans G_c ne nous permet pas de prédire quels sommets des triangles vont être recouverts par des arêtes ou des chaînes de longueur 2. Le pire cas étant de découper tous les triangles en chaînes de longueur 2, le nombre de chaînes de longueur 2 change en présence

[18] $\mathbb{1}_{\{Nb(TR)=0\}}$ est la fonction indicatrice qui indique la présence du premier slot lorsque il n'y a pas de triangle dans la solution optimale.

de triangles :

$$Nb^h(C_2) = \sum_{i(pair)=2}^{n_2-1} Nb(C_i) + Nb(TR)$$

La borne supérieure obtenue reste de la même forme générale :

$$C^h_{max} \leq 3n + 1 + max\{0, Nb(C_0) - Nb^h(C_1) - Nb^h(C_2)\} \tag{6.4}$$

Performance relative pour $\Pi_1^{PG} + TR$

Théorème 6.3.2 :
Il existe un algorithme en temps polynomial avec une performance relative bornée par $\frac{10}{9}$ pour le problème $\Pi_1^{PG} + TR$.

Preuve :
Comme précédemment, nous allons étudier la performance relative en fonction de $Nb(C_0)$. Le pire cas se produisant lorsque $Nb(C_0) = Nb(C_1) + \sum_{i=2}^{n_2-1}[(i-1)Nb(C_i)] + 3Nb(TR)$, nous obtenons en substituant $Nb(C_0)$ et n sur les bornes (6.3) et (6.4) :

- $C^{opt}_{max} \geq 3n + \mathbb{1}_{\{Nb(TR)=0\}} \geq 3n \geq 9Nb(C_1) + 18Nb(TR) + 6\sum_{i=2}^{n_2-1}[iNb(C_i)]$
- $C^h_{max} \leq 3n + 1 + Nb(C_0) - Nb^h(C_1) - Nb^h(C_2) \leq 3n + 1 + 2Nb(TR) + \frac{1}{2}\sum_{i=2}^{n_2-1}[iNb(C_i)]$

Le ratio de la performance relative est alors égale à :

$$\rho \leq \frac{C^h_{max}}{C^{opt}_{max}} \leq \frac{3n+1+2Nb(TR)+\frac{1}{2}\sum_{i=2}^{n_2-1}[iNb(C_i)]}{3n + \mathbb{1}_{\{Nb(TR)=0\}}} \leq 1 + \frac{1+2Nb(TR)+\frac{1}{2}\sum_{i=2}^{n_2-1}[iNb(C_i)]}{9Nb(C_1)+18Nb(TR)+6\sum_{i=2}^{n_2-1}[iNb(C_i)]}$$

$$\leq 1 + \frac{2Nb(TR)+\frac{6}{9}\sum_{i=2}^{n_2-1}[iNb(C_i)]}{18Nb(TR)+6\sum_{i=2}^{n_2-1}[iNb(C_i)]} \leq \mathbf{\frac{10}{9}}$$

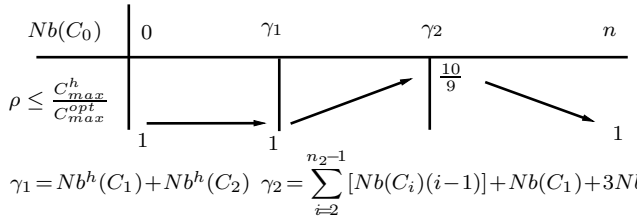


Fig. 6.12 – Variation de la performance relative $\boldsymbol{\rho}$ selon la valeur de $Nb(C_0)$

□

6.3.3 Approximation des problèmes Π_2^{PG} et Π_3^{PG}

L'étude de la complexité du problème Π_3^{PG} a été faite à la Section 6.2.2 page 89. En reprenant l'étude faite dans le Chapitre 4 avec un graphe de compatibilité quelconque et aucune tâche de traitement, il est possible d'étendre les approximations dans le pire des cas lorsque les tâches de traitement ont toutes un temps d'exécution égal à 1. L'ajout des tâches de traitement va permettre de combler les slots créés par les tâches d'acquisition, donc en reprenant les algorithmes définis aux Sections 4.3.2 page 54 et 4.3.3 page 59 et en les modifiant pour ajouter les tâches de traitement, nous pouvons dire sans perte de généralité que les ratios de performance relative seront au plus égaux à ceux trouvés au Chapitre 4. Pour le problème Π_3^{PG}, nous prendrons la pire des bornes de l'intervalle $\left[\frac{\mathbf{3}}{\mathbf{2}}, \frac{\mathbf{5}}{\mathbf{4}}\right]$ donné au Chapitre 4 pour Π_3, c'est-à-dire $\frac{\mathbf{3}}{\mathbf{2}}$.

6.4 Conclusion

Dans ce chapitre, nous étudions les problèmes d'ordonnancement sur mono processeur avec tâches d'acquisition en présence des deux contraintes précédemment étudiées, le graphe de compatibilité G_c et l'introduction des tâches de traitement. Nous étudions les mêmes problèmes que dans les Chapitre 4 et 5 toujours dans le but d'observer l'impact des contraintes sur l'évolution de la complexité des problèmes. Nous avons analysé les cas critiques se trouvant à cheval entre la polynomialité et la $\mathcal{NP}$-complétude selon la valeurs des paramètres. Puis nous avons comparé la complexité obtenue dans ce chapitre par rapport à celle d'Orman et Potts, ainsi que celle aux Chapitre 4 et 5.

Le chapitre est découpé en deux parties, la première portant sur l'étude de la complexité des problèmes critiques et par conséquent tous les problèmes plus généraux (resp. plus spécifiques), et la deuxième partie portant sur l'étude de l'approximation de ces mêmes problèmes.

Nous donnons trois preuves de $\mathcal{NP}$-complétude pour les problèmes notés Π_1^{PG}, Π_2^{PG} et Π_3^{PG}, et une preuve de polynomialité pour le problème Π_8^{PG} (Voir Figure 6.13). Les preuves sont basées sur le graphe de compatibilité G_c utilisant des réductions à partir de problèmes connus de recouvrement de sommets dans des graphes (PARTITION EN TRIANGLES, CLIQUE, COUVERTURE PAR DES CLIQUES, CHAÎNE HAMILTONIENNE). À partir de ces résultats nous en déduisons la $\mathcal{NP}$-complétude de tous les problèmes plus généraux.

Le premier constat que nous pouvons faire à la suite de ces résultats est que la contrainte d'incompatibilité entraîne la $\mathcal{NP}$-complétude des deux mêmes problèmes qu'au Chapitre 4, Π_1^{PG} et Π_2^{PG}.

Nous pouvons donc voir que **l'impact de la contrainte d'incompatibilité est plus forte que l'introduction des tâches de traitement**, qu'il y ait ou non les tâches de traitement, les problèmes Π_1^{PG} et Π_3^{PG} deviennent $\mathcal{NP}$-complets avec l'introduction de la contrainte d'incompatibilité. Toutefois, le fait que les problèmes Π_4^{PG}, Π_5^{PG}, Π_6^{PG} et Π_7^{PG} restent ouverts montre l'impact des tâches de traitement sur des problèmes critiques qui était polynomiaux avant l'introduction des tâches de traitement.

Les algorithmes en temps polynomial donnés sont également basés sur des couplages de taille maximum ou couplages parfaits de poids maximum. À partir de ces résultats nous en déduisons la polynomialité de tous les problèmes plus spécifiques.

La visualisation globale de la complexité des problèmes est représentée sur la Figure 6.13, nous avons mis en avant l'évolution de la complexité des problèmes lors de l'introduction de la contrainte d'incompatibilité et des tâches de traitement.

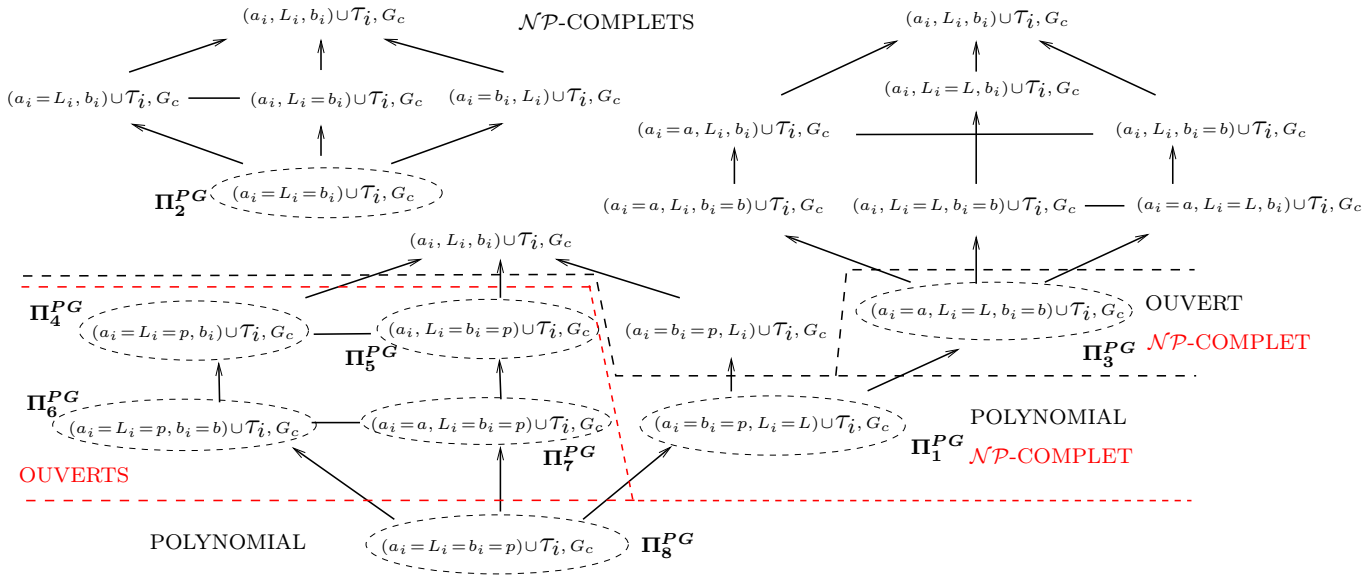


Fig. 6.13 – Visualisation globale de l'impact de l'introduction des tâches de traitement et de la contrainte d'incompatibilité sur la complexité des problèmes d'ordonnancement avec tâches d'acquisition. La ligne en pointillé noire représente les résultats sans la contrainte d'incompatibilité et sans tâches de traitement, et la ligne en pointillé rouge lorsque nous les introduisons.

Il est intéressant d'observer que la complexité des problèmes déjà $\mathcal{NP}$-complets avec l'introduction seule de la contrainte d'incompatibilité ne change pas avec l'introduction des tâches de traitement, les preuves deviennent plus difficiles mais restent sur le même schéma. Le plus gros changement a lieu sur les problèmes polynomiaux, les tâches de traitement rendant les problèmes plus difficiles, l'étude devient délicate et nous n'avons pas pu donner de résultat de complexité.

La deuxième partie sur l'approximation est basée sur les mêmes techniques que dans le Chapitre 4, néanmoins en s'intéressant à des cas bien particuliers comme pour le problème Π_1 lorsque $L = 2$, nous avons pu développer une technique de recouvrement des sommets de G_c par des arêtes et des chaînes de longueur 2 en temps polynomial. Cette technique permet d'obtenir de très bonnes bornes d'approximation à valeur constante.

Au terme de ce chapitre, nous pouvons observer l'importance de l'impact de la contrainte d'incompatibilité par rapport à l'introduction des tâches de traitement. Le graphe G_c change fondamentalement la structure des preuves, les différentes approches, et la manière d'ordonnancer les tâches d'acquisition. Le fait que les tâches de traitement soient préemptives explique aussi pourquoi ces tâches peuvent si bien s'adapter aux problèmes déjà existant, et donc avoir moins d'impact sur les résultats.

L'ensemble des résultats obtenus dans ce chapitre est récapitulé et référencé dans le tableau 6.2.

Problème	Complexité	Ratio d'approx.	Référence
$\Pi_1^{PG} : (a_i = b_i = p, L_i = L) \cup (\tau_i, pmnt), G_c$	$\mathcal{NP}$-complet	$\frac{(L+6)}{6}$	Page 87
$\Pi_2^{PG} : (a_i = L_i = b_i) \cup (\tau_i, pmnt), G_c$	$\mathcal{NP}$-complet	$\frac{3}{2}$	Page 85
$\Pi_3^{PG} : (a_i = a, L_i = L, b_i = b) \cup (\tau_i, pmnt), G_c$	$\mathcal{NP}$-complet	$\frac{3}{2}$	Page 89
$\Pi_8^{PG} : (a_i = L_i = b_i = p) \cup (\tau_i, pmnt), G_c$	Polynomial	1	Page 91
$\Pi_4^{PG} : (a_i = L_i = p, b_i) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 90
$\Pi_5^{PG} : (a_i, L_i = b_i = p) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 90
$\Pi_6^{PG} : (a_i = L_i = p, b_i = b) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 91
$\Pi_7^{PG} : (a_i = a, L_i = b_i = p) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 91

TAB. 6.2 – Résumé des résultats du Chapitre 6

Chapitre 7

Simulations

7.1 Introduction

En théorie de l'ordonnancement, comme dans beaucoup d'autres domaines, il est souvent difficile d'apprécier les résultats théoriques obtenus par différentes heuristiques. En effet, l'étude du pire des cas nous permet d'obtenir une borne supérieure pour un algorithme donné, mais en pratique il arrive que cette borne ne soit pas atteinte, voire approchée, selon les instances du problème étudié. Nous avons donc souvent recours à des expérimentations grandeur nature ou à des simulations pour pouvoir vérifier nos résultats et réaliser une meilleure analyse du comportement des différents algorithmes développés. Puisque notre problème vient du cas pratique de la torpille TAIPAN des roboticiens du LIRMM, il aurait été intéressant de réaliser des tests pratiques avec la torpille, ou du moins de récupérer des données pour réaliser des tests. Malheureusement, la torpille est encore en développement au sein du laboratoire et il a donc été impossible de pouvoir réaliser des tests grandeur nature. Nous avons donc été contraints de réaliser des simulations portant uniquement sur la vérification de l'efficacité des algorithmes proposés et l'observation de leur comportement selon la structure des données utilisées.

7.1.1 Création d'un simulateur

Nos algorithmes, donnés tout au long du manuscrit, portaient à chaque fois sur l'approximation ou la résolution de différents problèmes d'ordonnancement avec tâches-couplées. Certains des problèmes rencontrés n'ont même pas encore été étudiés d'un point de vu de l'approximation. La réalisation de ces tests a nécessité la création d'un simulateur et de différents générateurs de graphes aléatoires. Nous avons implémenté ce simulateur en C++, il permet de simuler tout type de problème rencontré dans la thèse selon la valeur des trois paramètres fondamentaux (a_i, L_i, b_i). La création d'un générateur de graphes aléatoires a été nécessaire afin de modéliser la contrainte d'incompatibilité, nous en avons créé plusieurs selon les structures particulières imposées au graphe G_c. Pour les problèmes qui ont été étudiés en détail et qui ont donné lieu au développement d'algorithmes, nous avons créé plusieurs classes d'objets génériques permettant de manipuler les différents algorithmes pour leur problème associé. Ce simulateur a également la possibilité d'afficher, sur la console ou dans un fichier, l'ordonnancement obtenu pour un problème et un algorithme donné. Cet affichage s'est révélé très utile pour la vérification des algorithmes et leur analyse.

Ce chapitre regroupe l'ensemble des tests réalisés sur les différents algorithmes développés. L'ajout des contraintes d'incompatibilité et de précédence étant tout nouveau pour ce type de problème en ordonnancement avec tâches-couplées, il nous est impossible de comparer nos résultats avec ceux existant dans la littérature. Nous allons donc donner des tableaux de résultats

pour chaque algorithme afin d'avoir des données de références. Nous avons choisi d'effectuer nos simulations sur des graphes aléatoires, générés à partir des deux paramètres suivants :

- n le nombre de sommets du graphe
- m le nombre d'arêtes du graphe

Nous tirons aléatoirement m arêtes dans une matrice de dimension $n \times n$. Pour certains problèmes particuliers, nous définissons un autre générateur créant des graphes à topologie particulière comme des bipartis complets par exemple. Chacune des sections suivantes contiendra l'ensemble des algorithmes donnés pour un chapitre en particulier portant sur l'étude de la complexité. Nous analyserons en moyenne le comportement du ratio de performance de chaque problème selon les résultats théoriques en faisant varier la densité du graphe de compatibilité et le nombre de tâches dans les instances simulées. Et nous analyserons également les temps de calculs pour obtenir les bornes supérieures, notons que les simulations ont été réalisées sur une station de travail équipée d'un processeur Intel Core 2 Duo 2.33 Ghz.

7.2 Prise en compte de la contrainte d'incompatibilité

Cette section regroupe les algorithmes 4, 5 et 6 donnés au Chapitre 4 où nous prenons en compte la contrainte d'incompatibilité. Selon les problèmes, nous proposons une visualisation du temps moyen d'exécution des algorithmes selon le nombre de sommets dans les instances expérimentées. Pour les algorithmes d'approximation nous calculons également les valeurs en moyenne des performances relatives obtenues. L'algorithme 4 n'a pas été implémenté comme les autres, en effet après la transformation du graphe G_c en G_c^m où certaines arêtes ont été retirées (voir Théorème 4.2.4 page 48), nous avons utilisé l'algorithme BLOSSOM IV développé par Cook et Rohe [24] afin de trouver un couplage parfait de poids minimum. Les temps de calculs pour résoudre notre problème dépendent essentiellement de BLOSSOM IV, puisque la transformation se fait en temps linaire, et il en est de même pour l'exécution des tâches également une fois que le choix des arêtes du couplage parfait a été fixé. Nous ne développerons donc pas les résultats expérimentaux, mais nous renvoyons les lecteurs vers le papier de Cook et Rohe [24] pour plus d'information.

7.2.1 Algorithme 5 pour le problème $\Pi_2 : 1|a_i = L_i = b_i, G_c|C_{max}$

Cette sous-section porte sur l'algorithme 5 donné page 55 permettant d'approximer le problème Π_2 avec une garantie de performance égale à $\frac{3}{2}$. Nous avons cherché à calculer les temps en moyenne d'exécution de l'algorithme, les bornes obtenues selon la densité du graphe G_c et le nombre de sommets le contenant. Dans le tableau 7.1, nous donnons les résultats expérimentaux obtenus par le simulateur, C_{max}^h XP (resp. ρ XP) représente la borne supérieure (resp. performance relative) obtenue en moyenne sur une vingtaine d'instances.

Les résultats présentés dans ce tableau montrent que notre performance relative théorique est proche de celles obtenues en simulation. La structure des tâches explique une telle proximité, en effet la technique des poupées russes nous ramène à des problèmes de remplissage de boites. Pour la plupart des instances rencontrées, nous pourrons souvent remplir ces boites sans trop de perte et ainsi obtenir une solution proche de l'optimal. Pour finir avec cet algorithme nous présentons sur la Figure 7.1 la courbe du temps moyen de l'algorithme selon le nombre de sommets dans le graphe G_c. Nous remarquons sur le tracé de la courbe que l'algorithme est exécutable en temps quasi-linéaire.

n	m	C^h_{max} XP	ρ théorique	ρ XP	Temps (ms)
25	30	$1483,65$	$1,5$	$1,41$	5
25	60	$1381,2$	$1,5$	$1,38$	5
25	90	$1371,45$	$1,5$	$1,37$	6
25	120	$1436,85$	$1,5$	$1,38$	4
50	60	$2935,05$	$1,5$	$1,42$	8
50	90	$2870,7$	$1,5$	$1,40$	6
50	120	$2893,5$	$1,5$	$1,39$	6
50	150	$2870,25$	$1,5$	$1,38$	7
100	130	$5826,3$	$1,5$	$1,41$	9
100	170	$5758,8$	$1,5$	$1,40$	9
100	230	$5913,6$	$1,5$	$1,40$	9
100	290	$5680,35$	$1,5$	$1,38$	9
150	190	$8686,95$	$1,5$	$1,41$	11
150	280	$8759,85$	$1,5$	$1,40$	12
150	370	$8597,4$	$1,5$	$1,39$	12
150	430	$8544,3$	$1,5$	$1,38$	12
200	250	$11681,5$	$1,5$	$1,41$	14
200	330	$11476,5$	$1,5$	$1,41$	13
200	450	$11395,3$	$1,5$	$1,39$	13
200	570	$11378,8$	$1,5$	$1,39$	13

TAB. 7.1 – Quelques résultats de la simulation de l'algorithme 5

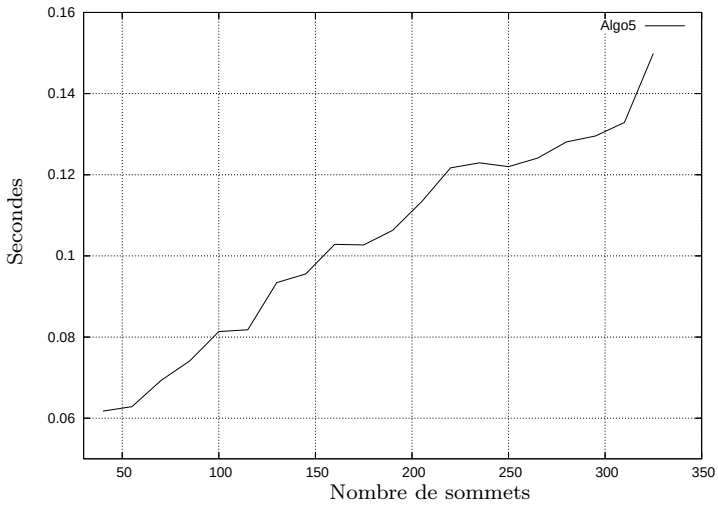


FIG. 7.1 – Illustration du temps d'exécution de l'algorithme 5 en fonction du nombre de sommets

7.2.2 Algorithme 6 pour le problème $\Pi_2' : 1|a_i = L_i = b_i, G_c = biparti\ complet|C_{max}$

Le problème étudié dans cette sous-section est une spécification du cas précédent, il porte sur l'algorithme 6 donné page 57 permettant d'approximer, avec une garantie de performance égale à $\frac{3}{2}$, le problème Π_2 lorsque le graphe G_c est un biparti complet. Comme précédemment, nous avons cherché à calculer le temps moyen d'exécution de l'algorithme, les bornes obtenues selon la densité du graphe $G_c^b = (X, Y, E)$ et le nombre de sommets le contenant. Dans ce cas particulier, nous avons fait varier le nombre de sommets dans G_c et celui dans l'ensemble $|X|$. Dans le tableau 7.2, nous donnons les résultats expérimentaux obtenus par le simulateur, C_{max}^h XP (resp. ρ XP) représente la borne supérieure (resp. performance relative) obtenue en moyenne sur une vingtaine d'instances.

n	$\|X\|$	C_{max}^h XP	ρ théorique	ρ XP	Temps (ms)
25	5	$978,6$	$1,16$	$1,01$	5
25	7	$1160,1$	$1,16$	$1,01$	6
25	9	$1428,3$	$1,16$	1	6
25	11	$1744,8$	$1,16$	1	6
75	15	$2908,2$	$1,16$	$1,01$	6
75	18	$31,93,35$	$1,16$	$1,02$	7
75	21	$3505,65$	$1,16$	$1,03$	7
75	24	$3870,9$	$1,16$	$1,01$	7
125	21	$4489,8$	$1,16$	$1,01$	8
125	27	$5069,7$	$1,16$	$1,01$	8
125	33	$5642,55$	$1,16$	$1,03$	10
125	39	$6252,45$	$1,16$	$1,01$	8
200	39	$7727,7$	$1,16$	$1,02$	10
200	49	$8670,3$	$1,16$	$1,03$	11
200	59	$9700,2$	$1,16$	$1,02$	13
200	69	$11088,9$	$1,16$	1	13

TAB. 7.2 – Quelques résultats de la simulation de l'algorithme 6

Les résultats présentés dans ce tableau montrent que notre performance relative théorique est éloignée de nos bornes supérieures obtenues en simulation. Comme nous l'avions dit à la Remarque 4.3.1 page 59, notre borne théorique a été largement majorés, et sur des instances qui semblent être le pire des cas, nous obtenions une borne de $\frac{8}{7}$, ce qui se rapproche de nos résultats expérimentaux. Pour finir avec cet algorithme nous présentons à la Figure 7.2 la courbe du temps moyen d'exécution de l'algorithme selon le nombre de sommets dans le graphe G_c. Nous observons sur le tracé de la courbe que l'algorithme est exécutable en temps quasi-linéaire.

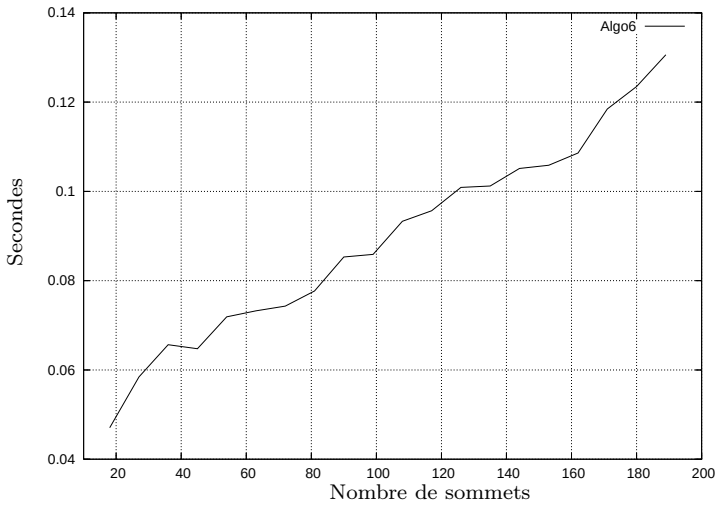


Fig. 7.2 – Illustration du temps d'exécution de l'algorithme 6 en fonction du nombre de sommets

Ceci finit la présentation des résultats obtenus lors des simulations effectuées pour les algorithmes du Chapitre 4. Nous allons maintenant voir celui du Chapitre 5.

7.3 Prise en compte des tâches de traitement

Cette section ne comporte qu'un algorithme puisque nous avons montré à la section 5.2.6 page 73 que les problèmes étudiés dans ce chapitre étaient liés à ceux sans tâches de traitement. Et ainsi l'approximation connue sans tâches de traitement nous avait permis de conjecturer sur l'évolution de la performance relative pour tout problème $\mathcal{NP} - complet$ de ce chapitre.

Le seul problème qui nous intéresse est le cas polynomial $\Pi_6^P : 1|(a_i = L_i = p, b_i) \cup (\tau_i, pmnt), G_c\ complet|C_{max}$ qui peut être résolu en temps polynomial par l'Algorithme 7. Nous avons montré que résoudre ce problème revenait à trouver un couplage maximum entre deux ensembles de sommets notés K et S selon la valeur des paramètres b_i et τ_i. Cette fois-ci le graphe de compatibilité G_c est complet, nous faisons varier le nombre de sommets dans le graphe et également la constante p. Plus p est grand et plus les temps d'inactivité créés par les tâches d'acquisition seront grands. Dans le tableau 7.3, nous donnons les résultats expérimentaux obtenus par le simulateur en appliquant l'Algorithme 7, C_{max}^h XP représente le makespan obtenu en moyenne sur une vingtaine d'instances.

Les résultats présentés dans ce tableau montrent que le temps moyen d'exécution de notre algorithme croit de manière logarithmique au fur et à mesure que nous augmentons le paramètre p. Pour finir avec cet algorithme nous présentons à la Figure 7.3 la courbe du temps moyen de l'algorithme selon le nombre de sommets dans le graphe G_c. Nous vérifions bien sur le tracé de la courbe que l'algorithme est exécutable en temps $O(nlog(n))$.

n	p	C^h_{max} XP	Temps (ms)
80	5	$1685,3$	6
170	5	$3591,7$	12
260	5	$5510,15$	26
350	5	$7350,5$	40
80	10	$2075,55$	8
170	10	$4408,7$	14
260	10	$6772,45$	27
350	10	$9044,35$	46
80	15	$2498,05$	6
170	15	$5250,1$	16
260	15	$8098,75$	29
350	15	$10849,5$	55

TAB. 7.3 – Quelques résultats de la simulation de l'algorithme 7

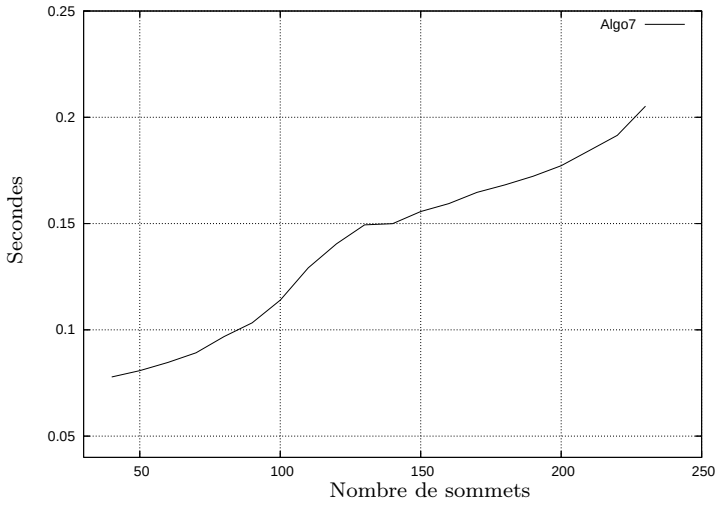


FIG. 7.3 – Illustration du temps d'exécution de l'algorithme 7 en fonction du nombre de sommets

Ceci finit la présentation des résultats obtenus lors des simulations effectuées pour le seul algorithme donné au Chapitre 5. Nous allons maintenant voir ceux du Chapitre 6.

7.4 Prise en compte des tâches de traitement et de la contrainte d'incompatibilité

Cette section regroupe les algorithmes 8 et 9 donnés au chapitre 6 où nous prenons en compte les contraintes d'incompatibilité et de précédence avec les tâches de traitement. Pour chaque algorithme implémenté, nous proposons une visualisation du temps moyen d'exécution selon le nombre de sommets dans les instances expérimentées. Pour les algorithmes d'approximation, nous calculons également les valeurs en moyenne des performances relatives obtenues.

7.4.1 Algorithme 8 pour $\Pi_1^{PG} : 1|(a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c|C_{max}$

Cette sous-section porte sur l'algorithme 8 donné page 93 permettant d'approximer le problème Π_1^{PG} avec une garantie de performance égale à $\frac{L+6}{6}$. Nous avons cherché à calculer le temps moyen d'exécution de l'algorithme, les bornes obtenues selon la densité du graphe G_c, le nombre de sommets le contenant et la valeur de la constante L. Dans le tableau 7.4, nous donnons les résultats expérimentaux obtenus par le simulateur, C_{max}^h XP (resp. ρ XP) représente la borne supérieure (resp. performance relative) obtenue en moyenne sur une vingtaine d'instances.

n	m	L	C_{max}^h XP	ρ théorique	ρ XP	Temps (ms)
100	120	4	387	1,66	1,29	14
100	160	4	366,6	1,66	1,22	35
100	200	4	358,2	1,66	1,19	63
100	120	8	620,75	2,33	2,06	19
100	160	8	577,65	2,33	1,92	39
100	200	8	562,3	2,33	1,87	65
150	160	4	596,35	1,66	1,32	20
150	200	4	569,55	1,66	1,26	29
150	240	4	548,6	1,66	1,21	70
150	160	8	954,7	2,33	2,12	19
150	200	8	900,35	2,33	2,01	29
150	240	8	864,6	2,33	1,92	54
200	220	4	789,25	1,66	1,32	30
200	260	4	760,75	1,66	1,27	32
200	300	4	735,8	1,66	1,23	65
200	220	8	1265,7	2,33	2,11	25
200	260	8	1214,85	2,33	2,02	32
200	300	8	1165,6	2,33	1,94	53

TAB. 7.4 – Quelques résultats de la simulation de l'algorithme 8

Les résultats présentés dans ce tableau montrent que notre performance relative théorique est assez proche de celles obtenues en simulation, mais c'est surtout dû au fait que nous avons pris des petites valeurs pour L. La borne théorique augmente rapidement en fonction de L , et remarquons que nous obtenons une borne égale à L lorsque L est immense. Pour ce qui est du temps d'exécution, la variation du paramètre L n'a que peu d'impact sur les résultats obtenus, seule la structure du graphe compte réellement. Pour finir avec cet algorithme nous présentons sur la Figure 7.4 la courbe du temps moyen d'exécution de l'algorithme selon le nombre de sommets dans le graphe G_c. Nous vérifions bien sur le tracé de la courbe que l'algorithme est exécutable en temps cubique.

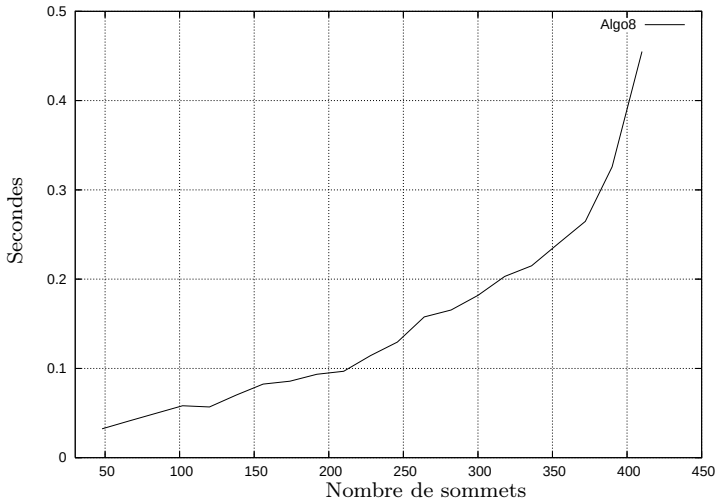


Fig. 7.4 – Illustration du temps d'exécution de l'algorithme 8 en fonction du nombre de sommets

7.4.2 Algorithme 9 pour $\Pi_1^{PG}:1|(a_i=b_i=1, L_i=2)\cup(\tau_i, pmtn), G_c|C_{max}$

Le problème étudié dans cette sous-section est une spécification du cas précédent, il nécessite plusieurs algorithmes définis dans le manuscrit. Tout d'abord il y a les algorithmes 1 et 2 pages 22 et 22 permettant d'obtenir un 2-recouvrement, puis l'algorithme 9 page 100 permettant d'approximer Π_1^{PG} avec une garantie de performance égale à $\frac{13}{12}$ sans présence de triangles (resp. $\frac{10}{9}$ en présence de triangles). Comme précédemment, nous avons cherché à calculer le temps moyen d'exécution de l'algorithme, les bornes obtenues selon la densité du graphe G_c et le nombre de sommets le contenant. Dans le tableau 7.5, nous donnons les résultats expérimentaux obtenus par le simulateur, C_{max}^h XP (resp. ρ XP) représente la borne supérieure (resp. performance relative) obtenue en moyenne sur une vingtaine d'instances où il y a des triangles.

Nous pouvons observer que nous sommes très proche d'une solution optimale pour notre performance relative, ceci vient du fait que les instances au pire des cas arrivent lorsqu'il y a un grand nombre de sommets isolés ou non saturés par le 2-recouvrement. Plus nous augmentons m et moins il y a de chance d'avoir des sommets non saturés, ce qui explique des temps de calculs plus rapide lorsque nous augmentons la densité u graphe G_c. Une visualisation du temps moyen d'exécution de l'algorithme 9 est donnée à la Figure 7.5 selon le nombre de sommets dans le graphe. La recherche d'un couplage maximum puis d'un 2-recouvrement entraîne une croissance rapide du temps d'exécution, et le tracé de la courbe montre que l'algorithme est exécutable en temps cubique.

n	m	C^h_{max} XP	ρ théorique	ρ XP	Temps (ms)
100	80	$304,75$	$1,11$	$1,02$	3
100	170	$301,6$	$1,11$	$1,01$	2
100	260	301	$1,11$	$1,01$	1
100	350	301	$1,11$	$1,01$	1
150	130	$455,95$	$1,11$	$1,02$	5
150	220	454	$1,11$	$1,01$	3
150	310	$451,3$	$1,11$	$1,01$	2
150	400	451	$1,11$	$1,01$	2
200	180	607	$1,11$	$1,02$	10
200	270	$602,65$	$1,11$	$1,01$	6
200	360	601	$1,11$	$1,01$	3
200	450	$601,3$	$1,11$	$1,01$	3
300	280	$910,9$	$1,11$	$1,01$	15
300	370	$906,4$	$1,11$	$1,01$	11
300	460	$902,5$	$1,11$	$1,01$	8
300	550	$901,9$	$1,11$	$1,01$	5
500	480	$1514,8$	$1,11$	$1,02$	32
500	570	$1510,6$	$1,11$	$1,01$	26
500	660	1505	$1,11$	$1,01$	23
500	750	$1503,7$	$1,11$	$1,01$	19

TAB. 7.5 – Quelques résultats de la simulation de l'algorithme 9

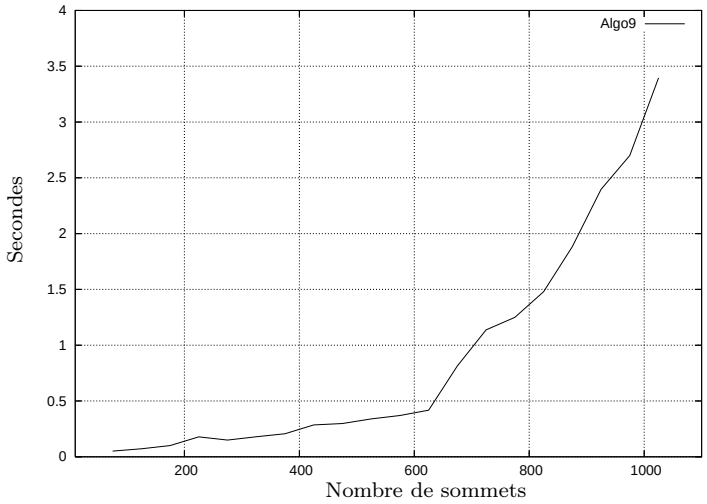


FIG. 7.5 – Illustration du temps d'exécution de l'algorithme 9 en fonction du nombre de sommets

Conclusion

Dans ce chapitre, nous avons cherché à valider les différents algorithmes développés tout au long du manuscrit, mais également à avoir un autre point de vu de l'impact de l'introduction de la contrainte de compatibilité et celle de précédence avec l'introduction des tâches de traitement. La première constatation, faite aux vues des résultats obtenus par nos simulations, est qu'il est difficile d'obtenir une performance relative théorique pertinente et proche des résultats expérimentaux. Pour la plupart des problèmes, les valeurs obtenues dans les colonnes ρXP sont souvent très inférieures à celles du ρ théorique calculé dans les chapitres précédents. Cette différence vient de la densité des graphes de compatibilité, passé un certain nombre d'arêtes, le graphe devient trop dense pour obtenir des instances proche du pire des cas. En effet nos approches portent pour la plupart sur des couplages maximum ou cliques maximales, et lorsque m augmente les possibilités de coupler les sommets ou de les recouvrir par des cliques maximales sont plus nombreuses.

Nous constatons également que l'introduction du graphe de compatibilité, et donc de certains problèmes de graphes, rend les algorithmes plus complexes et plus coûteux en temps. L'impact de cette contrainte était déjà visible dans les preuves de complexité dans les chapitres précédents, nous forçant à complètement changer les heuristiques classiques pour les problèmes d'ordonnancement avec tâches-couplées. D'un problème d'ordonnancement nécessitant des preuves de complexité basées sur la structure des tâches à exécuter, l'introduction de la contra inte d'incompatibilité nous a amené à avoir des preuves basées sur des problèmes de recouvrement dans un graphe quelconque.

Sur les différents tableaux de résultats donnés dans ce chapitre, nous avons pu constater que le temps d'exécution des algorithmes pouvait fortement varier pour certains problèmes lorsque nous augmentions le nombre d'arêtes dans le graphe, que ce soit dans un sens ou dans l'autre. En effet parmi les algorithmes testés, seul l'algorithme 9 devient plus rapide en augmentant le nombre d'arêtes dans le graphe, ceci vient du fait que nous cherchons à minimiser les sommets isolés. Donc la densité du graphe G_c va jouer sur l'efficacité et la rapidité des algorithmes.

Au final nous pouvons conclure sur le fait que les simulations ont permis de vérifier l'efficacité et la justesse de nos résultats théoriques, elles ont également permis de mettre en avant l'impact de l'introduction du graphe de compatibilité sur les temps de calculs et les résultats obtenus, ce qui rejoint nos observations et conclusions à propos des résultats théoriques.

Conclusion et perspectives

Les travaux présentés dans ce manuscrit ont porté sur l'étude théorique de la complexité et de l'approximation pour des problèmes d'ordonnancement avec tâches-couplées avec prise en compte ou non du graphe de compatibilité et des tâches de traitement. Nous avons chercher à classifier la plupart des problèmes d'ordonnancement avec tâches-couplées selon les valeurs des trois paramètres fondamentaux (a_i, L_i, b_i). Cette étude est motivée par un problème de robotique portant sur une torpille sous-marine autonome nommée TAIPAN. Cette dernière doit remplir plusieurs objectifs dans un certain laps de temps, ces objectifs nécessitent l'exécution de différentes tâches, certaines d'acquisition et d'autres de traitement.

Les tâches dites d'acquisition nécessitent l'utilisation de différents capteurs se trouvant sur la torpille, selon les types de mesures certains capteurs seront soumis à des interférences. En effet l'acquisition de données se faisant par propagation d'onde, des interférences au moment de la réception des données peuvent fausser les résultats. Dans le but de gérer ce type de problèmes, un graphe de compatibilité entre les tâches d'acquisition est produit permettant de connaître à l'avance les tâches susceptibles d'être exécutées en parallèle.

La présentation et la modélisation du problème de la torpille TAIPAN a fait l'objet du Chapitre 3 de la Partie I de ce mémoire. Dans un premier temps, à partir d'une problématique issue de la robotique, il fut nécessaire de modéliser au sens de la théorie de l'ordonnancement au mieux ce problème. Nous nous sommes appuyés sur des résultats (voir même démontrer de nouveaux) de la théorie des graphes afin de représenter au mieux les différentes tâches et contraintes rencontrées. À partir de cette modélisation, un état de l'art sur les problèmes avec tâches-couplées, représentées par trois paramètres fondamentaux (a_i, L_i, b_i), nous a permis de positionner notre problème par rapport aux travaux existants. Les résultats de complexité les plus importants ont été réalisés par Orman et Potts [57], ils ont étudié la complexité de la plupart des problèmes d'ordonnancement sur mono processeur avec tâches-couplées en faisant varier les paramètres fondamentaux (a_i, L_i, b_i). Ils donnent une représentation en treillis des résultats de complexité obtenus (voir Figure 3.5 à la page 33).

Nos travaux se sont inspirés et calqués sur ces treillis afin d'avoir une base théorique nous permettant de mieux percevoir l'impact des différentes contraintes rencontrés, celle d'incompatibilité et de précédence avec les tâches de traitement .

Nos principales contributions se trouvent dans la Partie II du manuscrit, cette partie est divisée en trois chapitres. Le premier prend en compte l'introduction du graphe de comptabilité entre les tâches d'acquisition, nous avons donné la complexité de chaque problème dans la Section 4.2 à la page 42. Nous avons montré l'évolution des preuves pour l'étude de la complexité des différents problèmes rencontrés, en effet l'introduction du graphe de compatibilité entraîne une approche différente pour les preuves de $\mathcal{NP}$-complétude et de polynomialité, nécessitant l'utilisation de la théorie des graphes. Ces résultats montrent bien l'impact de la contrainte d'incompatibilité sur les résultats existants d'Orman et Potts. Dans la Section 4.3 à la page 51 nous avons donné plusieurs algorithmes d'approximation avec garantie de performance

pour les principaux problèmes rencontrés, il s'agit des problèmes les plus critiques d'après les treillis obtenus pour l'étude de la complexité. La présence d'un graphe de compatibilité quelconque induit une difficulté supplémentaire, en effet la plupart des problèmes nécessitent des recouvrements de cardinalité minimum par des cliques ou des chaînes de longueurs différentes. Ces problèmes étant bien connus dans la littérature pour être non-approximables, les seules approches réalisables en temps polynomial se basent sur des couplages maximum ou des couplage parfaits de poids maximum.

Le deuxième chapitre prend en compte l'introduction cette fois-ci des tâches de traitement, de la même manière que dans le chapitre précédent, nous avons donné la complexité de chaque problème dans la Section 5.2 à la page 68. Nous avons ainsi montré que la complexité obtenue par Orman et Potts ne changait pas lorsque nous introduisions les tâches de traitement, toutefois les preuves de $\mathcal{NP}$-complétude et les algorithmes polynomiaux utilisés pour résoudre certains problèmes sont plus complexes et nécessitent une analyse plus fine que celles faites par Orman et Potts. L'étude de l'approximation de ces problèmes n'a pas fait lieu d'une étude approfondie puisque les problèmes sans tâches de traitement devraient être plus simple à approximer une fois les tâches introduites grâce à la préemptivité de celles-ci. Nous laissons ouvert cette étude mais proposons tout de même en conjecture une variation de la performance relative pour chaque problème selon les tâches de traitement (voir Section 5.3 page 81).

Enfin le dernier chapitre de cette partie regroupe les deux contraintes prises en compte dans les deux chapitre précédents. Nous avons à nouveau donné la complexité de chaque problème rencontré lorsque cela était possible dans la Section 6.2 page 86. Nous avons constaté que le graphe de compatibilité avait plus d'impact aux vus des preuves de $\mathcal{NP}$-complétude, l'introduction des tâches de traitement ne modifie que légèrement les preuves faites au Chapitre 4. Toutefois l'intérêt de la prise en compte des tâches de traitement se manifestent sur les problèmes qui étaient polynomiaux aux Chapitres 4 et 5. En effet, la présence des tâches de traitement nous amène à gérer plusieurs critères, ce qui nous empêche d'avoir ne serait-ce qu'une intuition de la forme d'une solution optimale pour certains problèmes. A la suite d'une longue étude sur ces cas critiques, nous proposons en conjecture différents résultats de complexité pour ces problèmes ouverts. Pour ce qui est de l'étude de l'approximation de ces problèmes, nous proposons plusieurs algorithmes d'approximation avec garantie de performance pour chaque problème critique se trouvant à la limite de la $\mathcal{NP}$-complétude et de la polynomialité selon les treillis obtenus (Figure 3). L'une d'entre elles utilise un 2-recouvrement qui consiste à saturer un maximum de sommets par des chaînes de longueur au plus 2, nous donnons plusieurs définitions, propriétés et résultats pour le 2-recouvrement dans la Section 2.3.1 page 16. Les tâches de traitement permettent d'obtenir de meilleurs ratios d'approximation grâce à la préemptivité des tâches.

L'ensemble des problèmes étudiés dans cette deuxième partie est regroupé dans les trois Tableaux 1, 2 et 3, et une vue d'ensemble est donnée sur les treillis aux Figures 1, 2 et 3. Ces travaux ont apporté une vue d'ensemble des résultats théoriques de complexité et d'approximation pour les problèmes d'ordonnancement avec tâches-couplées selon les contraintes ajoutées Ces résultats permettront d'avoir une base théorique solide pour toutes études ultérieures.

La fin de la deuxième partie du manuscrit porte sur les résultats expérimentaux des différents algorithmes développés dans les Chapitres 4, 5, 6. L'étude sur l'approximation, des différents problèmes qui a été réalisée, portait sur une analyse au pire cas, il est donc intéressant de voir ce qu'il se passe en moyenne selon la densité du graphe de compatibilité, et les paramètres pris en compte. L'implémentation a été réalisée en C++, où nous avons développé un simulateur

gérant les différents problèmes selon les paramètres pris en compte et les contraintes présentes. Ce chapitre permet de mieux apprécier l'efficacité de certains algorithmes et de comparer les résultats expérimentaux obtenus par les heuristique par rapport aux bornes théoriques calculées préalablement.

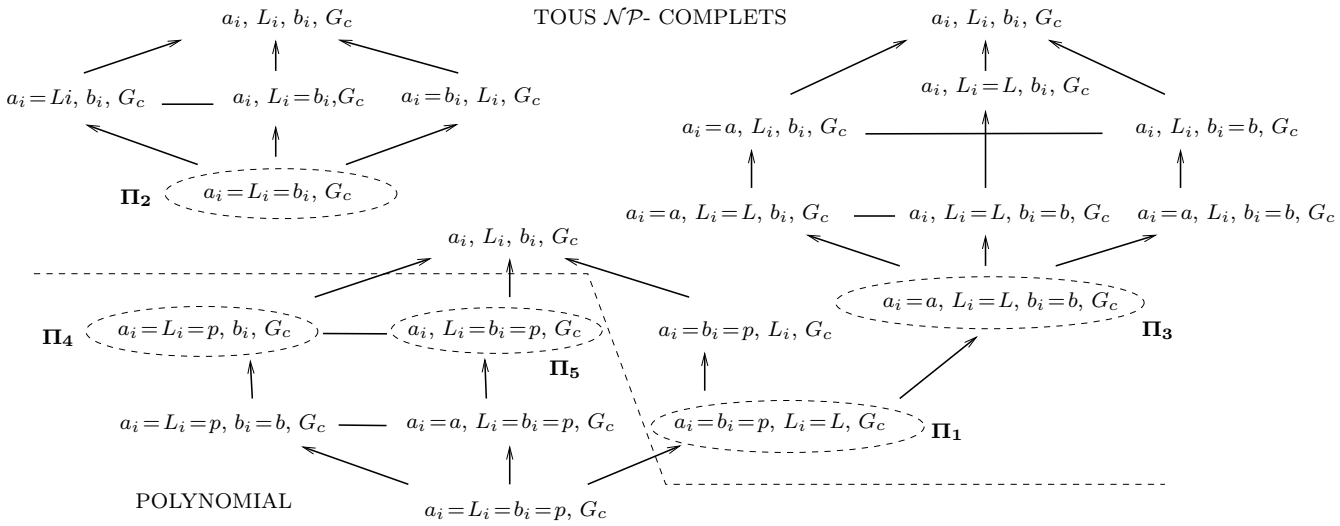


Fig. 1 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition en présence de contrainte d'incompatibilité

Problème	Complexité	Ratio d'approx.	Référence
$\Pi_1 : (a_i = b_i = p, L_i = L), G_c$	$\mathcal{NP}$-complet	$\frac{7p+L}{4p}$	Page 52
$\Pi_2 : (a_i = L_i = b_i), G_c$	$\mathcal{NP}$-complet	$\frac{3}{2}$	Page 56
$\Pi_3 : (a_i = a, L_i = L = a+b, b_i = b), G_c$	$\mathcal{NP}$-complet	$\left[\frac{3}{2}, \frac{5}{4}\right]$	Page 59
$\Pi_4 : (a_i = L_i = p, b_i), G_c$	Polynomial	1	Page 47
$\Pi_5 : (a_i, L_i = b_i = p), G_c$	Polynomial	1	Page 50

Tab. 1 – Résumé des résultats du Chapitre 4

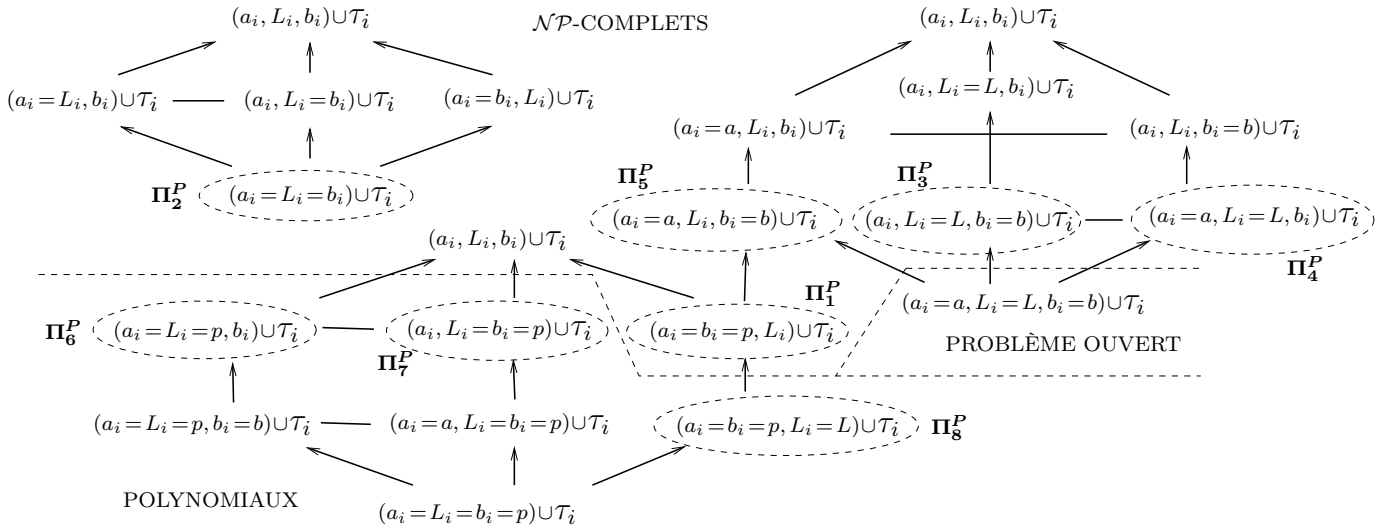


Fig. 2 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition en présence de contrainte de précédence

Problème	Complexité	Référence
$\Pi_1^P : (a_i = b_i = p, L_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 69
$\Pi_2^P : (a_i = L_i = b_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 72
$\Pi_3^P : (a_i, L_i = L, b_i = b) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 72
$\Pi_4^P : (a_i = a, L_i = L, b_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 72
$\Pi_5^P : (a_i = a, L_i, b_i = b) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	$\mathcal{NP}$-complet	Page 73
$\Pi_6^P : (a_i = L_i = p, b_i) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	Polynomial	Page 73
$\Pi_7^P : (a_i, L_i = b_i = p) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	Polynomial	Page 80
$\Pi_8^P : (a_i = b_i = p, L_i = L) \cup (\mathcal{T}_i, pmnt), G_c$ *complet*	Polynomial	Page 80

Tab. 2 – Résumé des résultats du Chapitre 5

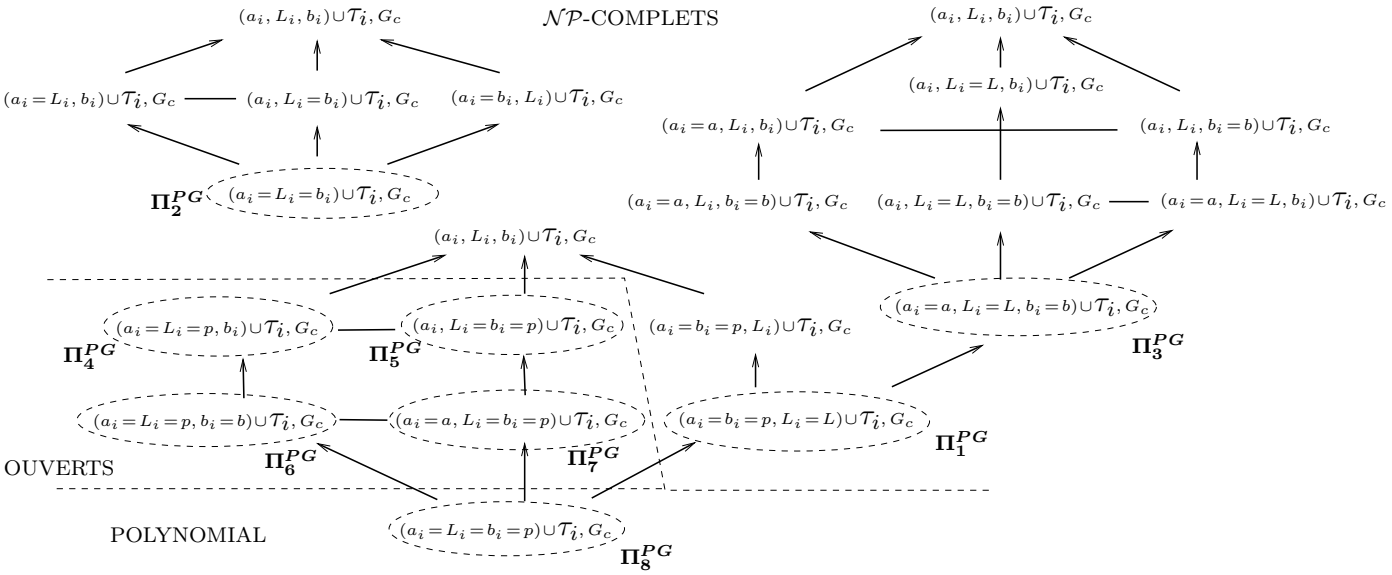


Fig. 3 – Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition et tâches de traitement en présence de contraintes de précédence et d'incompatibilité

Problème	Complexité	Ratio d'approx.	Référence
$\Pi_1^{PG} : (a_i = b_i = p, L_i = L) \cup (\tau_i, pmnt), G_c$	$\mathcal{NP}$-complet	$\frac{(L+6)}{6}$	Page 87
$\Pi_2^{PG} : (a_i = L_i = b_i) \cup (\tau_i, pmnt), G_c$	$\mathcal{NP}$-complet	$\frac{3}{2}$	Page 85
$\Pi_3^{PG} : (a_i = a, L_i = L, b_i = b) \cup (\tau_i, pmnt), G_c$	$\mathcal{NP}$-complet	$\frac{3}{2}$	Page 89
$\Pi_8^{PG} : (a_i = L_i = b_i = p) \cup (\tau_i, pmnt), G_c$	Polynomial	1	Page 91
$\Pi_4^{PG} : (a_i = L_i = p, b_i) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 90
$\Pi_5^{PG} : (a_i, L_i = b_i = p) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 90
$\Pi_6^{PG} : (a_i = L_i = p, b_i = b) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 91
$\Pi_7^{PG} : (a_i = a, L_i = b_i = p) \cup (\tau_i, pmnt), G_c$	Ouvert		Page 91

Tab. 3 – Résumé des résultats du Chapitre 6

Perspectives

Les travaux, qui ont été commencés en fin de thèse et qui vont se poursuivre après celle-ci, vont porter sur une approche stochastique et en temps-réel du problème général de la torpille. Nous comparerons cette stratégie avec les résultats théoriques obtenus préalablement. Une telle étude permettrait d'étudier le problème du point de vue des roboticiens, qui doivent prendre en compte le fait que la torpille est autonome, qu'elle doit exécuter des groupes de tâches de manière périodique, ou encore gérer différents problèmes qui peuvent survenir en temps réel durant sa mission.

Tous les paramètres et les contraintes entrant en jeux vont être forcément modélisés de manière aléatoire, c'est la raison pour laquelle une étude stochastique du problème général de la torpille nous intéresse. En nous basant sur les résultats des chapitres précédents, nous pouvons chercher à voir si les problèmes qui étaient $\mathcal{NP}$-complets pour l'étude déterministe deviennent plus faciles à étudier en ajoutant de l'aléatoire sur certains paramètres. L'objectif de nos travaux ayant été la minimisation du makespan, nous allons devoir étudier la complexité des mêmes problèmes lorsque l'objectif est la minimisation de la somme des temps de complétude (ce qui revient à minimiser la moyenne de la fin d'exécution de chaque tâche : $\sum_j C(j)$). À partir de ces résultats théoriques, il sera intéressant de comparer les résultats expérimentaux en prenant compte de la nature aléatoire de certains paramètres.

Nous avons déjà commencé à réaliser une telle étude, en donnant une nouvelle modélisation du problème de la torpille plus proche de la réalité. A partir de cette modélisation, nous avons étudié les différents problèmes obtenus précédemment en rendant aléatoire certains paramètres fondamentaux et en laissant les graphes de précédence ou de compatibilité fixés dès le départ, vu que nous en avons une connaissance globale. Nous analyserons les règles d'ordonnancement efficaces pour ces problèmes où la torpille reçoit des groupes de tâches, dont les paramètres suivent différentes lois de distribution.

Une deuxième modélisation plus précise consisterait à prendre en compte le côté périodique de l'exécution des tâches. Envisageons que la torpille communique avec un bateau qui lui envoie des groupes de tâches à réaliser périodiquement. Chacun de ces groupes de tâches à exécuter (représentant les objectifs à réaliser pour la torpille d'un point de vu robotique) va arriver aléatoirement et devra être pris en compte par la torpille en temps réel, chaque groupe aura sa propre période d'exécution. Lorsque la torpille ne pourra pas gérer l'exécution de tous les groupes de tâches envoyés par le bateau, elle devra abandonner certains groupes. L'objectif est de trouver les meilleures règles à appliquer garantissant en moyenne les meilleurs ordonnancements qui permettent d'exécuter un maximum de groupes de tâches périodiquement, et développer des algorithmes d'approximation robustes et efficaces.

Par la suite, une étude intéressante consisterait à travailler sur une architecture multiprocesseur. L'objectif serait d'analyser la robustesse des algorithmes d'approximation et l'évolution des ratios de performance. Enfin pour finir, nous pourrons étudier l'introduction simultanée de ces deux contraintes, et ainsi avoir des tâches périodiques sur une architecture multiprocesseur.

Annexe A

Preuves pour le Chapitre 5

Introduction

Ce court chapitre regroupe les deux preuves de $\mathcal{NP}$-complétude pour les problèmes Π_2^P : $1|prec, (a_i = b_i = L_i) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$ et $\Pi_3^P : 1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$ du Chapitre 5 sections 5.2.2 et 5.2.3 page 72. Les deux preuves étant similaires à celle du problème Π_1^P, elle même déjà basée sur les preuves d'Orman et Potts, nous avons préféré les laisser en Annexe.

A.1 Étude de $\Pi_2^P : 1|prec, (a_i = b_i = L_i) \cup (\tau_i \geq 1, pmtn), G_c\ complet|C_{max}$

Étudions le cas spécifique $\Pi_2^{P'} : 1|prec, (a_i = b_i = L_i) \cup (\tau_i = 1, pmtn), G_c\ complet|C_{max}$, où toutes les tâches de traitement ont un temps d'exécution égal à 1. Si ce cas est $\mathcal{NP}$-complet alors le cas général Π_2^P sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3.

Théorème A.1.1 :
Le problème de décider si une instance de notre problème $\Pi_2^{P'}$ possède un ordonnancement optimal d'une certaine longueur y est $\mathcal{NP}$-complet.

Preuve :
Nous allons montrer que le problème $\mathcal{NP}$-complet 3-PARTITION (Problème 5.2.1) se réduit vers le problème de décision $\Pi_2^{P'}$. À partir d'une instance de 3-PARTITION où $Q = \{e_1, e_2, \ldots, e_{3q}\}$, nous construisons l'instance suivante du problème de décision $\Pi_2^{P'}$. L'instance comprend $n = 4q$ tâches d'acquisition (resp. de traitement) telles que :

$$a_i = L_i = b_i = 2e_i qE^2 + i \text{ et } \tau_i = 1 \qquad \text{pour } i = 1, \ldots, 3q$$
$$a_i = L_i = b_i = 6qE^3 + 27q + (i - 3q) \text{ et } \tau_i = 1 \qquad \text{pour } i = 3q + 1, \ldots, 4q$$

Existe t-il un ordonnancement valide avec $C_{max} \leq y = 3(6q^2E^3 + 27q^2 + q(q-1)/2) + 1$?

$\Rightarrow$ Nous allons montrer que s'il existe une solution au problème 3-Partition, alors $\Pi_2^{P'}$ admet une solution avec $C_{max} \leq y$. Nous supposons qu'il existe une solution au problème 3-PARTITION, ce qui entraîne qu'il existe des sous-ensembles Q_j pour $j = 1, \ldots, q$ avec $\sum_{i \in Q_j} e_i = E$. Les tâches de partition correspondant à Q_j sont exécutées durant le slot d'inactivité de la tâche de division A_{3q+j}, où $j = 1, \ldots, q$ et $Q_j = \{A_u, A_v, A_w\}$, comme indiqué dans

la Figure A.1. Les q tâches de division sont exécutées consécutivement sans chevauchement ni temps d'inactivité entre elles.

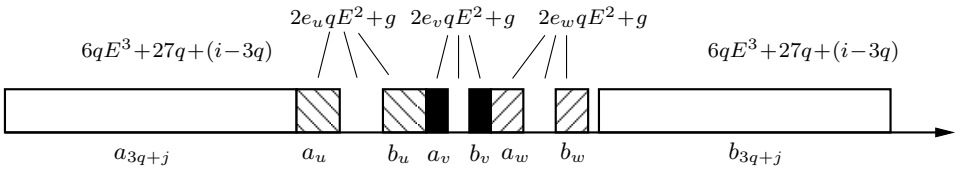


FIG. A.1 – Ordonnancement de la tâche A_{3q+j} et des tâches comprises dans Q_j

Après l'exécution de toutes les tâches d'acquisition, il reste encore les tâches de traitement à exécuter. Soit R_{3q+j} le temps d'inactivité entre la fin de l'exécution de la troisième tâche A_w et le début de l'exécution de la seconde sous-tâche b_{3q+j}. Nous avons alors :

$$\begin{aligned}
R_{3q+j} &= L_{3q+j} - (p_{a_u} + p_{a_u} + L_u) - (p_{a_v} + p_{a_v} + L_v) - (p_{a_w} + p_{a_w} + L_w) \\
&= (6qE^3 + 27q + j - 3q) - (6e_uqE^2 + 3u) - (6e_vqE^2 + 3v) - (6e_wqE^2 + 3w) \\
&= (6qE^3 + 27q + j - 3q) - 6qE^3 - 3u - 3v - 3w \ , \quad \text{puisque } \sum_{i\in Q_j} e_i = E \\
&= 27q + (j - 3q) - 3u - 3v - 3w \\
&\geq 27q + (3q + 1 - 3q) - 3(3q) - 3(3q - 1) - 3(3q - 2) = 10 \qquad \text{(A.1)}
\end{aligned}$$

Donc $R_{3q+j} = 10$, ce qui entraîne que les tâches de traitement τ_u, τ_v, τ_w peuvent s'exécuter dans ce slot d'inactivité, ainsi que la tâche de traitement τ_{3q+j-1} qui vient de la tâche de division précédente A_{3q+j-1}. Toute les tâches de traitement sont exécutées sans changer la longueur de l'ordonnancement excepté pour la tâche de traitement de la dernière tâche de division A_{4q}.Le makespan est alors égal à :

$$C_{max} = \sum_{j=3q+1}^{4q} (p_{a_j} + L_j + p_{b_j}) + 1 = 3(6q^2E^3 + 27q^2 + q(q+1)/2) + 1 = y \qquad \text{(A.2)}$$

Donc il existe bien un ordonnancement valide pour le problème $\Pi_2^{P'}$ avec $C_{max} \leq y$.

$\Leftarrow$ Nous allons montrer que s'il existe une solution au problème d'ordonnancement $\Pi_2^{P'}$ avec $C_{max} \leq y$, alors il existe une solution au problème 3-PARTITION.

Tout d'abord, notons que la dernière tâche exécutée dans l'ordonnancement ne peut être qu'une tâche de traitement, ce qui entraîne que nous allons analyser l'exécution des autres tâches avec un makespan de longueur $y - 1 = 3(6q^2E^3 + 27q^2 + q(q + 1)/2)$. Remarquons également que les tâches de division ne peuvent avoir leur exécution qui se chevauchent dû au fait que chacune de ces tâches a une taille différente. Donc leur exécution consécutive donne un temps de processus égal à $\sum_{j=3q+1}^{4q}(a_j + L_j + b_j) = y - 1$. L'exécution des tâches de division utilisant $y - 1$ unités de temps, les tâches de partition et de traitement sont forcément exécutées dans les slots d'inactivité des tâches de division.

Notons que les exécutions des tâches de partition ne peuvent se chevaucher dû fait que chaque tâche a une taille différente. De plus, soit Q_j l'ensemble des tâches de partition ordonnancées dans le slot d'inactivité L_{3q+j} de la tâche de division A_{3q+j}, pour $j = 1, \ldots, q$.

Supposons maintenant que l'ensemble des Q_j ne donne pas de solution au problème 3-PARTITION. Alors, il existe une tâche de division A_{3q+k} telle que $\sum_{i\in Q_k} e_i \geq (E + 1)$, et

ainsi la somme des durées d'exécution des tâches de partition contenues dans l'ensemble Q_k correspondant est :

$$\begin{aligned}
\sum_{A_i \in Q_k} (a_i + L_i + b_i) &= \sum_{A_i \in Q_k} 3(2e_i qE^2 + i) \\
&= (\sum_{A_i \in Q_k} e_i) 6qE^2 + 3 \sum_{A_i \in Q_k} i \\
&\geq (E+1)6qE^2 \\
&= 6qE^3 + 6qE^2 \\
&> 6qE^3 + 27q + k \qquad \text{car } k \leq q \text{ et } E \geq 3 \\
&= L_{3q+k}
\end{aligned}$$

Donc les tâches de partition contenues dans Q_k ne peuvent s'ordonnancer dans le slot d'inactivité L_{3q+k}, et par conséquent $\sum_{i \in Q_j} e_i \leq E$ pour tout $j = 1, \ldots, q$. De plus, nous savons que s'il existe un ensemble Q_k tel que $\sum_{A_i \in Q_k} e_i < E$, alors il existe un ensemble Q_l tel que $\sum_{A_i \in Q_l} e_i > E$. Ce qui entraîne que $\sum_{A_i \in Q_j} e_i = E$ pour tout $j = 1, \ldots, q$, et par conséquent $Q_1, \ldots, Q_q$ représentent une 3-PARTITION.

Nous avons donc 3-PARTITION $\leq_T \Pi_2^{P'}$, et nous savons que 3-PARTITION (2.2.9) est $\mathcal{NP}$-complet. Ainsi, d'après le Lemme 2.1.2 nous pouvons conclure que le problème $\Pi_2^{P'}$ est $\mathcal{NP}$-complet.

□

A.2 Étude de $\Pi_3^P : 1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c\ complet|C_{max}$

Étudions le cas spécifique $\Pi_3^{P'} : 1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i = 1, pmtn), G_c\ complet|C_{max}$, où toutes les tâches de traitement ont un temps d'exécution égal à 1. Si ce cas est $\mathcal{NP}$-complet alors le cas général Π_3^P sera $\mathcal{NP}$-complet d'après le Lemme 2.1.3.

Théorème A.2.1 :
Le problème de décider si une instance de notre problème $\Pi_3^{P'}$ *possède un ordonnancement optimal d'une certaine longueur y est* $\mathcal{NP}$*-complet.*

Preuve :
Cette preuve est basée sur la même structure que la précédente. Nous allons montrer que le problème $\mathcal{NP}$-complet 3-PARTITION se réduit vers le problème de décision $\Pi_3^{P'}$. À partir d'une instance de 3-PARTITION où $Q = \{e_1, e_2, \ldots, e_{3q}\}$, nous construisons en temps polynomial l'instance suivante du problème de décision $\Pi_3^{P'}$. L'instance comprend $n = 4q$ tâches d'acquisition (resp. de traitement) telles que :

$$\begin{aligned}
(a_i, L_i, b_i) = (e_i, E, b) \text{ et } \tau_i = 1 \qquad &\text{pour } i = 1, \ldots, 3q, \\
(a_i, L_i, b_i) = (E+1, E, b) \text{ et } \tau_i = 1 \qquad &\text{pour } i = 3q+1, \ldots, 4q, \\
\text{où } b = min_{i=1,\ldots,3q}\{e_i\} - 1.&
\end{aligned}$$

Existe t-il un ordonnancement valide avec $C_{max} \leq y = q(3E + 1 + b) + 1$?

⇒ De la même manière que précédemment nous allons montrer que trouver une solution au problème 3-PARTITION, implique que $\Pi_3^{P'}$ admet une solution avec $C_{max} \leq y$. Nous supposons qu'il existe une solution au problème 3-PARTITION, ce qui entraîne qu'il existe des sous-ensembles Q_j pour $j = 1, \ldots, q$ avec $\sum_{i \in Q_j} e_i = E$. Les tâches de partition correspondant à $Q_j = \{A_u, A_v, A_w\}$ ont leurs premières sous-tâches a_u, a_v, a_w qui sont exécutées durant le slot d'inactivité de la tâche de division A_{3q+j}, où $j = 1, \ldots, q$, comme indiqué dans la Figure A.2. Les secondes sous-tâches b_u, b_v, b_w sont exécutées après la seconde sous-tâche b_{3q+j} de la tâche de division A_{3q+j}. L'ordonnancement d'une tâche de division avec ses tâches de partition forment ce que nous appellerons par la suite un bloc. L'exécution des q blocs obtenus sans temps d'inactivité entre eux donne le makespan total.

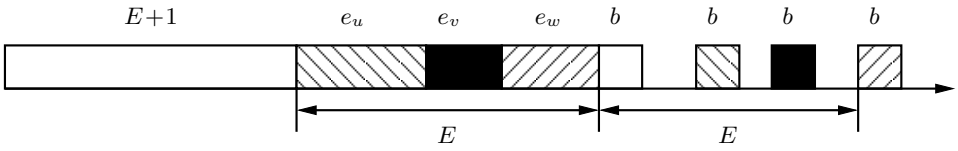


Fig. A.2 – Ordonnancement des tâches de partition de Q_j dans la tâche de division A_{3q+j}

Soit R_{3q+j} le slot d'inactivité de chaque bloc, remarquons que $R_{3q+j} = E - 3b \geq 4$ pour $j = 1, \ldots, q$, car $b = min_{i=1,\ldots,3q}\{e_i\} - 1$ et tous les e_i sont différents. Donc avec $R_{3q+j} \geq 4$, nous pouvons exécuter dans ce slot les tâches de traitement τ_u, τ_v, τ_{3q+j} et une autre tâche de traitement qui vient d'une des tâches de partition du bloc précédent. Donc toutes les tâches de traitement sont exécutées sans changer la longueur du makespan, excepté pour la tâche de traitement de la dernière tâche de partition du dernier bloc.

Les q blocs contribuent chacun à $(3E + 1 + b)$ à la valeur objective, et nous avons alors :

$$C_{max} = q(3E + 1 + b) + 1 = y \tag{A.3}$$

Donc il existe bien une solution au problème $\Pi_3^{P'}$ avec $C_{max} \leq y$

⇐ Nous allons montrer que s'il existe une solution au problème d'ordonnancement $\Pi_3^{P'}$ avec $C_{max} \leq y$, alors il existe une solution au problème 3-PARTITION.

Tout d'abord, notons que la dernière tâche exécutée dans l'ordonnancement ne peut être qu'un tâche de traitement, ce qui entraîne que nous allons analyser l'exécution des autres tâches avec un makespan de longueur $y - 1 = q(3E + 1 + b)$. De plus, du fait que $a_i > L_j$ pour tout i et j dans $\{3q + 1, \ldots, 4q\}$, les tâches de division ne peuvent entrecroiser leur exécution entre elles. De la même manière, $a_i > L_j$ pour $i \in \{3q + 1, \ldots, 4q\}$ et $j \in \{1, \ldots, 3q\}$. Donc, la seule manière d'entrecroiser l'exécution d'une tâche de division et d'une tâche de partition est de mettre la tâche de division en premier.

Montrons par l'absurde que la seule manière d'ordonnancer les tâches de partition et de division est de former q blocs avec l'exécution d'une tâche de division A_{3q+j} entrelacée avec trois tâches de partition contenues dans l'ensemble $Q_j = \{A_u, A_v, A_w\}$ telles que $\sum_{A_i \in Q_j} e_i = E$.

Supposons donc que l'ensemble des Q_j ne donne pas de solution au problème 3-PARTITION. Alors il existe une tâche de partition, noté A_s, dont l'exécution ne se chevauche pas avec celle d'une tâche de division. Nous avons q blocs avec $(3q - 1)$ tâches de partition exécutées dans le slot des tâches de division, et donc q blocs ayant chacune un temps d'exécution égal à $(2E + \sum_{A_i \in Q_j} e_i + b + 1)$, avec $j = 1, \ldots, q$. Sans prendre en compte la tâche A_s ni les tâches de

traitement, la somme totale des temps d'exécution des blocs est égale à :

$$\begin{aligned} & q(2E+1+b) + \sum_{j=1}^{q} \sum_{A_i \in Q_j} e_i \\ = \ & q(2E+1+b) + qE - e_s \\ = \ & q(3E+1+b) - e_s \end{aligned} \tag{A.4}$$

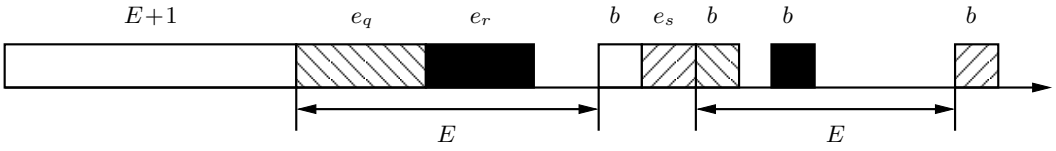


Fig. A.3 – Ordonnancement d'une tâche A_s hors d'un bloc

Le temps restant pour exécuter la tâche de partition A_s est égal à e_s, et le temps séquentiel de cette tâche vaut $e_s + b$. Donc il faut faire chevaucher A_s avec d'autres tâches de partition sans chevaucher une tâche de division. Dans le meilleur des cas, nous pouvons exécuter la sous-tâche a_s juste après la fin d'exécution de la sous-tâche b_{3q+k} d'une tâche de division A_k. Mais avec cette configuration le temps d'exécution du bloc, créé par A_k et l'ensemble Q_k, est égal à $(3E+2b+e_s+1)$ où $Q_k = \{e_q, e_r, e_s\}$ (see figure A.3). Ainsi le makespan total est égal à la somme du temps d'exécution des $q-1$ blocs, tels que $|Q_j| = 3$ avec $j = 1, \ldots, q-1$, et le bloc k :

$$\begin{aligned} C_{max} &= [(q-1)(2E+b+1) + \sum_{j=1}^{q-1} \sum_{A_i \in Q_j} e_i] + 3E + 2b + e_s + 1 \\ &= q(2E+B+1) + qE + E + b - e_q - e_r \\ &\geq q(3E+b+1) + b, \quad \text{car } E \geq e_r + e_q \end{aligned} \tag{A.5}$$

Par conséquent, si nous considérons une tâche de partition qui ne chevauche pas son exécution avec celle d'une tâche de division, il n'existe pas de solution pour Π_2 avec $C_{max} \leq y$. Les tâches de partition doivent donc être chevauchées avec une tâche de division dans le but d'obtenir un ordonnancement valide. De plus, nous savons que s'il existe un ensemble Q_k tel que $\sum_{A_i \in Q_k} e_i < E$, alors il existe également un ensemble Q_l tel que $\sum_{A_i \in Q_l} e_i > E$. Ce qui entraîne que $\sum_{A_i \in Q_j} e_i = E$ pour $j = 1, \ldots, q$, et par conséquent l'ensemble des $Q_1, \ldots, Q_q$ défini une 3-PARTITION.

□

Bibliographie

[1] A. A. Ageev and A. V. Kononov. Approximation algorithms for scheduling problems with exact delays. In *WAOA*, pages 1–14, 2006.

[2] D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled-tasks. *Mathematical Methods of Operations Research*, 59 :193–203(11), June 2004.

[3] S. R. Arikati and C. P. Rangan. Linear algorithm for optimal path cover problem on interval graphs. *Information Processing Letters*, 35(3) :149–153, 1990.

[4] G. Aussiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexité et approximation : Combinatorial optimization problems and their approximability properties*. Springer Verlag, November 1999.

[5] K. R. Baker. *Introduction to Sequencing and Scheduling*. New York : Wiley, USA, 1974.

[6] K. R. Baker and Z.-S. Su. Sequencing with due dates and early start times to minimize maximum tardiness. *Naval Research Logistics Quarterly*, 21 :171–176, 1974.

[7] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps, and nonapproximability-towards tight results. *SIAM Journal on Computing*, 27(3) :804–915, 1998.

[8] C. Berge. Two Theorems in Graph Theory. *Proceedings of the National Academy of Science*, 43 :842–844, September 1957.

[9] F. Berman, D. S. Johnson, F. T. Leighton, P. W. Shor, and L. Snyder. Generalized planar matching. *Journal of Algorithms*, 11(2) :153–184, 1990.

[10] P. Berman and M. Karpinski. 8/7-approximation algorithm for (1,2)-tsp. In *SODA '06 : Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 641–648, New York, NY, USA, 2006. ACM.

[11] J. Blażewicz, K. Ecker, T. Kis, C.N. Potts, M. Tanas, and J. Whitehead. Scheduling of coupled tasks with unit processing times. *Technical report, Poznan University of Technology*, 2009.

[12] J. Blażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling computer and manufacturing processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[13] M. A. Bonuccelli and D. P. Bovet. Minimum node disjoint path covering for circular-arc graphs. *Information Processing Letters*, 8(4) :159–161, 1979.

[14] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest starts and due date constraints. *Naval Research Logistics Quarterly*, 18 :511–517, 1971.

[15] P. Bratley, M. Florian, and P. Robillard. On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/g/fmax)$ problem. *Naval Research Logistics Quarterly*, 20 :57–67, 1973.

[16] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Potts, and J. Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers & OR*, 36(2) :301–307, 2009.

[17] R.L. Brooks. On colouring nodes of a network. *Proceedings of the Cambridge Philosophical Society*, 37 :194–197, 1941.

[18] J. Carlier. The one-machine sequencing problem. *European Journal of Operations Research*, 11 :42–47, 1982.

[19] G. J. Chang and D. Kuo. The l(2, 1)-labeling problem on graphs. *SIAM Journal on Discrete Mathematics*, 9(2) :309–316, 1996.

[20] F. T. Boesch S. Chen and B. McHugh. On covering the points of a graph with point-disjoint paths. *Graphs and Combinatorics*, 406 :201–212, 1974.

[21] Ph. Chrétienne and C. Picouleau. Scheduling with communication delays : a survey. In *Scheduling theory and its applications*, pages 641–648. John Wiley & sons, 1995.

[22] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of scheduling*. Addison-Wesley publishing compagny, Reading, Massachusetts, 1967.

[23] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[24] W. Cook and A. Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing (INFORMS)*, 11(2) :138–148, 1999.

[25] J. Edmonds. Maximum matching and a polyhedron with $0, 1$ vertices. *Journal of Research the National Bureau of Standards*, 69 B :125–130, 1965.

[26] A. A. Ageev et A. E. Baburin. Approximation algorithms for the simple and two-machine scheduling problems with exact delays. *to appear in Operations Research Letters*, 2006.

[27] L. Finta and Z. Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70(3) :247–266, 1996.

[28] M. Florian, P. Trepant, and G. McMahon. An implicit enumeration algorithm for the machine sequencing problem. *Management Sciences*, 17 :B782–B792, 1971.

[29] H. N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *Journal of the ACM*, 23(2) :221 – 234, 1976.

[30] Harold N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *Journal of the ACM*, 23(2) :221 – 234, 1976.

[31] 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.

[32] M. R. Garey, D. S. Johnson, B. B. Simon, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and desdlines. *SIAM Journal Computer*, 10 :256–269, 1981.

[33] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3) :237–267, 1976.

[34] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press Inc, 1980.

[35] S. E. Goodman and S. T. Hedetniemi. On the hamiltonian completion problem. *Graph Theory and Combinatorics*, pages 262–272, 1973.

[36] S. E. Goodman, S. T. Hedetniemi, and P. J. Slater. Advances on the hamiltonian completion problem. *Journal of the ACM*, 22(3) :352–360, 1975.

[37] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[38] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1) :19–23, 1993.

[39] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM Journal Computation*, 27(4) :1142–1167, 1998.

[40] A. El Jalaoui. *Gestion contextuelle de tâches pour le contrôle d'une véhicule sous-marin autonome*. PHD, LIRMM, Décembre 2007.

[41] Z. Jin and X. Li. On the k-path cover problem for cacti. *Theoretical Computer Science*, 355(3) :354–363, 2006.

[42] R.M. Karp. Reducibility among combinatorial problems. *J.W. Thatcher (Eds.)*, pages 85–103, 1972.

[43] D. G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *STOC*, pages 240–245, 1978.

[44] S. Kundu. A linear algorithm for the hamiltonian completion number of a tree. *Information Processing Letters*, 5 :55–57, 1976.

[45] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. *Preemptive scheduling of uniform machines subject to release dates*. in : W. R. Pulleyblank, Progress in Combinatory Optimization, New York, 1984.

[46] B. J. Lageweg, J.K. Lenstra, and A.H.G. Rinnooy Kan. Minimizing maximum lateness on one machine : Computational experience and some applications. *Statistica Neerlandica*, 30 :25–41, 1976.

[47] E. L Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sciences*, 19 :544–546, 1973.

[48] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. *Recent developments in deterministic sequencing and scheduling : A survey*. in Deterministic and stochastic scheduling, Dempster, Lenstra et Rinnooy Kan (eds.), Reidel, Dordrecht, 1982.

[49] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26 :22–35, 1978.

[50] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annale Discrete Mathematics*, 1 :343–362, 1977.

[51] R. Lin, S. Olariu, and G. Pruesse. An optimal path cover algorithm for cographs. *Computers & Mathematics with Applications*, 30 :75–83, 1995.

[52] S. Masuyama and T. Ibaraki. Chain packing in graphs. *Algorithmica*, 6(6) :826–839, 1991.

[53] J. Misra and R. E. Tarjan. Optimal chain partitions of trees. *Information Processing Letters*, 4(1) :24–26, 1975.

[54] S. Moran and Y. Wolfstahl. Optimal covering of cacti by vertex-disjoint paths. *Theoretical Computer Science*, 84 :179–197, 1988.

[55] A. J. Orman, C.N. Potts, A. K. Shahani, and A. R. Moore. Scheduling for a multifunction phased array radar system. *European Journal of Operations Research*, 90 :13–25, 1996.

[56] A. J. Orman, A. K. Shahani, and A. R. Moore. Modelling for the control of a complex radar system. *Computers & OR*, 25(3) :239–249, 1998.

[57] A.J. Orman and C.N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72 :141–154, 1997.

[58] V. Th. Paschos. *Complexité et approximation polynomiale*. Lavoisier, Hermes, 2004.

[59] C. N. Potts. An algorithm for the single machine sequencing problem with precedence constraints. *Mathematic Programming study*, 13 :78–87, 1980.

[60] C. N. Potts. A lagrangian based branch and bound algorithm for a single machine sequencing with precedence constraints to minimize total weighted completion time. *Operations Research*, 28 :1436–1441, 1980.

[61] R.D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27 :477–481, 1980.

[62] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal Computer*, 12(2) :294–299, 1983.

[63] Z. Skupien. Path partitions of vertices and hamiltonicity of graphs. *Procesing of second Czechoslovakian symposium on Graph Theory, Prague*, 1974.

[64] W. E Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3 :59–66, 1956.

[65] R. Srikant, R. Sundaram, K. S. Singh, and C. P. Rangan. Optimal path cover problem on block graphs and bipartite permutation graphs. *Theoretical Computer Science*, 115(2) :351–357, 1993.

[66] G. Steiner. On the k-path partition problem in cographs. *Cong. Numer.*, 147 :89–96, 2000.

[67] G. Steiner. On the k-path partition of graphs. *Theoretical Computer Science*, 290 :2147–2155, 2003.

[68] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. Ser. 2*, 42 :230–265, 1936.

[69] A. M. Turing and J.-Y. Girard. La machine de turing. *Seuil, Paris,France*, 1995. Translated from English by Julien Basch et Patrice Blanchard.

[70] J.-H. Yan and G. J. Chang. The path-partition problem in block graphs. *Information Processing Letters*, 52(6) :317–322, 1994.

[71] J.-H. Yan, G. J. Chang, S. M. Hedetniemi, and S. T. Hedetniemi. k-path partitions in trees. *Discrete Applied Mathematics*, 78(1-3) :227–233, 1997.

Table des figures

Liste des tableaux

Liste des publications

Conférences internationales avec comité de lecture

[1] G. Simonin, R. Giroudeau, J.-C. König, "Extended matching problem for a coupled-tasks scheduling problem". Dans **TMFCS'09** (Theoretical and Mathematical Foundations of Computer Science), Juillet 2009. Orlando, Floride, USA.

[2] G. Simonin, R. Giroudeau, J.-C. König, "Complexity and approximation for scheduling problem for a torpedo". Dans **CIE'39** (Computers & Industrial Engineering), **IEEE**, Juillet 2009. Troyes, France.

[3] G. Simonin, B. Darties, R. Giroudeau, J.-C. König, "Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor". Dans **MISTA'09** (Multidisciplinary International Scheduling conference : Theory & Applications), Aout 2009. Dublin, Irlande.

[4] G. Simonin, R. Giroudeau, J.-C. König, "Polynomial-time algorithms for scheduling problem for coupled-tasks in presence of treatment tasks". Dans **ISCO 2010** (International Symposium on Combinatorial Optimization), Mars 2010. Hammamet, Tunisie.

[5] G. Simonin, R. Giroudeau, J.-C. König, "Complexity and approximation for scheduling problem for coupled-tasks in presence of compatibility tasks". Dans **PMS 2010** (Project Management and Scheduling), Avril 2010. Tours, France.

Conférences nationales avec comité de lecture

[6] G. Simonin, R. Giroudeau, J.-C. König , "Complexité et approximation pour un problème d'ordonnancement avec tâches couplées". Dans **RenPar'18** (Rencontres francophones du Parallélisme), Février 2008. Fribourg, Suisse.

[7] G. Simonin, "Étude de la complexité de problèmes d'ordonnancement avec tâches-couplées sur monoprocesseur". Dans **MajecStic2008** (Manifestation des Jeunes Chercheurs en Sciences et Technologies de l'Information et de la Communication), Octobre 2008. Marseille.

[8] G. Simonin, A.-E. Baert, A. Jean-Marie, R. Giroudeau, "Problème d'acquisition de données par une torpille". Dans **ROADEF'09** (Recherche Opérationnelle et d'Aide à la Décision), Février 2009. Nancy.

Résumé : Les travaux présentés dans cette thèse portent sur l'étude de la complexité et de l'approximation des problèmes d'ordonnancement en présence de tâches-couplées sur un mono-processeur. Ces problèmes sont motivés par la modélisation d'un problème de robotique portant sur une torpille sous-marine d'exploration. Cette torpille a pour objectif d'exécuter deux types de tâches : celles d'acquisition et celles de traitement. Les tâches d'acquisition sont semblables à des tâches-couplées, et les tâches de traitement sont des tâches classiques. La torpille utilise différents capteurs pour réaliser les acquisitions, certains capteurs ne peuvent pas être utilisés en même temps pour cause d'interférences. Nous introduisons donc un graphe de compatibilité permettant de représenter les tâches d'acquisition pouvant avoir leurs exécutions qui se chevauchent. La torpille possède un monoprocesseur embarqué permettant d'exécuter toutes la tâches.

La première partie de nos travaux s'intéresse à la modélisation du problème, aux différentes tâches utilisées et aux contraintes qui leur sont appliquées. Nous mettons en avant l'impact de la contrainte de compatibilité, nous forçant à utiliser la théorie des graphes pour analyser nos problèmes. Enfin, nous finissons cette partie avec un état de l'art sur les différents résultats portant sur l'ordonnancement de tâches-couplées sur mono-processeur, et sur des problèmes de recouvrement de sommets dans des graphes. Dans une seconde partie, nous donnons la classification des problèmes possibles en faisant varier les paramètres des tâches-couplées. Nous donnons des preuves de complexité pour certains problèmes se trouvant à la limite entre la polynomialité et la $\mathcal{NP}$-complétude selon les valeurs des paramètres. Pour chaque problème $\mathcal{NP}$-complet, nous proposons des algorithmes d'approximation en temps polynomial et analysons les bornes obtenues selon les paramètres ou les topologies du graphe de compatibilité. L'ensemble des résultats est décomposé en trois chapitres prenant chacun en compte l'introduction d'une contrainte (d'incompatibilité et/ou de précédence). Tout au long de cette partie nous cherchons à montrer l'impact de l'introduction de la contrainte d'incompatibilité sur la complexité des problèmes d'ordonnancement avec tâches-couplées, à travers les preuves de $\mathcal{NP}$-complétude et les techniques employées pour résoudre ou approximer un problème.

Abstract : The last couple of years have seen the advent of sub-marine machines like the sub-marine torpedo. This torpedo execute two kind of tasks : acquisition tasks and treatment tasks. The acquisition tasks are equivalent to coupled-tasks, and treatment tasks are equivalent to classical tasks with preemption allowed. The torpedo possess several captors in order to realize the acquisitions, few captors cannot be used in same time because of the interferences. In this way, we introduce a compatibility graph in order to represent the acquisition tasks which can be executed in the same time. The torpedo possess a mono-processor used to execute the different tasks.

In the first part, we introduce the model of the problem, and define the different tasks and their constraints. We discuss on the impact of the compatibility constraint on the general problem. Lastly, we finish this part with a related works on general scheduling theory, on coupled-tasks, and on the different cover problems in the graph theory. In a second part, we give the classification of the different problems according to the parameters of the coupled-tasks. We give several proofs of complexity for specific problem which are at the limit between polynomiality and completeness. For each studied problem, we propose either a optimal solution with an algorithm in polynomial time, or an approximation algorithm with guarantee of performance. For few problems, we study the complexity in details according to specific parameters or different topologies for the compatibility graph. All the long of this part, we try to show the impact of the introduction of the compatibility constraint on the scheduling problem with coupled-tasks.

Discipline : Informatique
Mots Clés : Ordonnancement, Tâche couplée, Complexité, Approximation, Graphe.

LIRMM UMR 5506 - 161 rue Ada, 34392 Montpellier Cedex 5 - France