ordonnancement et communications

Rodolphe Giroudeau

Académie de Montpellier
Université Montpellier II
Sciences et Techniques du Languedoc

Habilitation à Diriger des Recherches

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

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

Ordonnancement et Communications

par **Rodolphe GIROUDEAU**

Date de soutenance : le 19 octobre 2012
Composition du jury

Messieurs	*Vangelis Paschos*	Rapporteurs
	Christophe Picouleau	
	Denis Trystram	
Madame	*Alix Munier-Kordon*	Présidente du Jury
Monsieur	*Stéphan Thomassé*	Examinateur
Monsieur	*Jean-Claude König*	Examinateur

Remerciements

Je tiens à remercier les membres de mon jury de thèse :

Alix Munier-Kordon pour l'honneur qu'elle m'a fait de présider ce jury.

Chrisitophe Picouleau, *Vangelis Paschos* et *Denis Trystram* pour avoir accepté d'être les rapporteurs de cette volumineuse habilitation. C'est pour moi une grande joie, de savoir que des chercheurs de cette qualité aient apprécié mon travail.

Je tiens à remercier *Jean-Claude König* et *Stéphan Thomassé* d'avoir acceptés d'être membre du jury.

Je tiens à exprimer ma plus profonde gratitude à *Jean-Claude König*, avec qui j'ai beaucoup de plaisir à travailler, qui m'a initié à la recherche et qui s'est toujours intéressé à mon travail. *Jean-Claude* est plus qu'un collègue.

J'adresse mes remerciements à tous les membres du Laboratoire pour leur aide précieuse au quotidien.

Je voudrai exprimer mon amour à ma maman *Catherine Kermarrec*, tout simplement car c'est ma maman.

Je tiens à mentionner les noms de *Bernard Lemoine, le daron* et de *Franck Cardarelli* pour ce qui m'ont apporté sur les terrains et à l'extérieur.

Je tiens à exprimer ma sincère amitié à mon frère *Gaël Thomas* pour les heures passées sur les terrains de sports et en dehors. C'est un plaisir de t'avoir comme frère.

Je voudrai pour finir exprimer mon amour à *Séverine, Romane et Maëlle* pour m'avoir encouragé dans les moments délicats, d'être présentes au quotidien, de m'écouter et de m'aimer tel que je suis.

À ma grand-mère Jeanne,

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Première partie

Présentation générale et rappels

CHAPITRE

1 Présentation

1.1 Introduction

Il est communément admis que les problèmes d'ordonnancement s'intéressent à l'allocation (optimale) des différentes parties d'une application, représentée par un graphe de précédence $G = (V, E)$ qui caractérise les contraintes chronologiques entre ces différentes parties, aux ressources/machines, afin que l'application soit accomplie le plus rapidement et/ou au moindre coût. Cette définition est très générique et permet de couvrir un large spectre de problèmes d'optimisation combinatoire. Nous pouvons citer comme domaines d'application : ordonnancement de projet (via la méthode PERT), les problèmes liés aux ateliers de production, lignes d'assemblage, les systèmes d'exploitation des ordinateurs, les systèmes distribués et systèmes embarqués, les problèmes d'emplois du temps, des problèmes d'affectation, de transports, de tournées, ...

Quoique récente (début des années 1960), la théorie de l'ordonnancement a fait l'objet d'études très poussées, tant au niveau de la complexité, que de la recherche de solutions exactes ou de solutions approchées avec ou sans garantie de performance [1]. De nombreux ouvrages spécifiques et dédiés aux problèmes d'ordonnancement ont été publiés. Nous pouvons référencer trois livres classiques portant sur l'étude des problèmes d'ordonnancement en général [16], [99], [133].

Je me suis intéressé aux problèmes d'ordonnancement lorsque j'ai commencé ma thèse de Doctorat sous la direction de Evripidis Bampis en 1996 à l'université d'Évry Val d'Essonne dont le sujet portait sur « L'impact des délais de communications hiérarchiques sur la complexité et l'approximation des problèmes d'ordonnancement ». J'ai poursuivi l'étude de ces problèmes depuis mon intégration dans l'équipe APR (« Algorithme et Performances

[1] Ces notions seront précisées dans le chapitre 2

des Réseaux ») au sein du LIRMM. Ce manuscrit établit une synthèse de mes travaux depuis quelques années.

1.2 Organisation de la thèse

Le plan de cette thèse comporte trois grandes parties (de tailles inégales). La première partie porte sur la présentation générale de ce manuscrit, un rappel sur la théorie de la complexité et de l'approximabilité et pour finir les problèmes d'ordonnancement. La suivante se focalise sur les problèmes d'ordonnancement pour le parallélisme. La dernière concerne des applications potentielles pour deux problèmes d'ordonnancement.

Au **chapitre 2** nous procéderons à un rappel sur la théorie de la complexité et de l'approximabilité. Nous y présenterons les bases telles que la notion de transformation polynomiale, les classes de complexité ($\mathcal{P}$,$\mathcal{NP}$ et $\mathcal{NP}$-complet). Nous illustrons via deux exemples la nécessité de poursuivre, de compléter la classification en utilisant la théorie de l'approximation. Nous donnerons le théorème de l'impossibilité qui permet de déterminer un seuil d'approximation pour un problème d'optimisation Π à partir d'un résultat de $\mathcal{NP}$-complétude pour une variante décisionnelle de Π. Nous rappellerons également le théorème du « gap » qui permet la détermination d'un seuil d'approximation k pour un problème d'optimisation Π à partir d'un seuil d'approximation k' pour un problème d'optimisation Π'. Dans la littérature cette réduction est appelée la *gap*-réduction. Nous présenterons également les classes d'approximation dépendant de l'instance.

Nous listerons les problèmes $\mathcal{NP}-$complets qui seront utilisés dans la suite lors des différentes transformations polynomiales proposées.

Au **chapitre 3** les définitions fondamentales, les hypothèses et les concepts, en théorie de l'ordonnancement, qui seront utilisés dans la suite de ce manuscrit seront décrits. Nous commencerons par présenter l'ensemble des tâches. Ensuite, nous décrirons via deux modèles de base en ordonnancement statique (le système multiprocesseur et les processeurs dédiés), l'environnement des processeurs et l'influence sur les caractéristiques des tâches. Pour finir, nous énumérons les critères d'optimalité (ou fonctions objectifs) classiques en théorie de l'ordonnancement. Un exemple viendra illustrer les différents critères d'optimalité.

Après, nous énumérons les voies possibles et les stratégies pour tenter de résoudre au « mieux » (notion qui sera formalisée dans ce chapitre) les problèmes dont la complexité des meilleurs algorithmes connus est exponentielle. Nous détaillerons la notation synthétique (usuelle) à trois champs $\alpha|\beta|\gamma$.

Les chapitres suivants concernent les problèmes d'ordonnancement pour le parallélisme. Le but de l'ordonnancement pour le parallélisme est d'obtenir des modèles de plus en plus fiables et réalistes tout en conservant des résultats théoriques exploitables. Les modèles faisant abstraction de la hiérarchie mémoire ne peuvent être utilisés dans les architectures modernes, et ainsi la notion de communication doit être prise en compte.

Le **chapitre 4** abordera les résultats de non-approximabilité et d'approximation dans le cas du modèle homogène en présence des grands délais de communication et nous redémontrerons le résultat classique pour le problème $UET - UCT$, en utilisant une transformation à partir d'une variante de $3SAT$. Le modèle avec communication homogène est un modèle très largement étudié dans les années 90. Ce modèle capture la notion de délai de communication potentielle entre deux tâches i et j, soumises à une contrainte de précédence. Les contraintes de précédence entre les tâches sont données par un graphe de précédence. Le délai n'est effectif que si les deux tâches i et j s'exécutent sur deux processeurs différents. Dans ce chapitre nous présenterons, une classification tant en complexité qu'en approximation, des principaux résultats connus sur ce modèle.

Au **chapitre 5**, nous présenterons un modèle avec des communication hiérarchiques dans lequel plusieurs niveaux de communications sont pris en compte. Ce modèle tente de modéliser les architectures où les processeurs sont regroupés sous forme de grappes de processeurs. Une grappe sera constituée de deux ou plusieurs processeurs reliés par un réseau totalement connecté. Les grappes seront également reliées entre-elles par un réseau totalement connecté. Avec ces caractéristiques, nous pouvons hiérarchiser les délais de communications entre les tâches soumises à des contraintes de précédence. En effet, nous distinguons les communications inter-grappes et les communications intra-grappes. Ce modèle est une généralisation du modèle à communications homogènes. Nous étudierons ce modèle au niveau de la complexité et de l'approximation, et nous étudierons également les stratégies et les techniques qui résistent aux passages du modèle homogène au modèle hiérarchique.

Le **chapitre 6** sera dédié à la présentation d'un modèle qui prend en compte le placement des tâches sur les processeurs comme une donnée essentielle. Ce modèle se caractérise par le fait que dans la plupart des modèles d'ordonnancement étudiés le graphe de processeurs (sur lesquels les tâches s'exécutent) est implicitement supposé comme étant complet (tous les processeurs sont totalement connectés). Dans ce chapitre, nous relaxons cette hypothèse sur l'environnement des processeurs en considérant plusieurs graphes structurés (chaîne, hypercube, ...). Et nous souhaitons mesurer l'impact de

l'introduction des réseaux de processeurs sur la complexité et l'approximation des problèmes d'ordonnancement.

De même, plusieurs résultats de complexité et d'approximation selon les hypothèses seront développés.

Le **chapitre 7** portera sur des applications. Dans un premier temps, nous étudierons du point de vue de la complexité et sur le développement de solutions algorithmiques pour un problème d'ordonnancement appliqué à l'acquisition de données pour une torpille en immersion. Dans ce cadre, nous utiliserons le modèle des tâches-couplées sur un monoprocesseur en présence d'un graphe de compatibilité. Les tâches-couplées est un modèle pour lequel les tâches sont constituées de deux sous-tâches avec un délai de communication entre ces sous-tâches indilatable et incompréssible (dans la littérature, nous retrouvons ce modèle avec la terminologie "communication exacte"). La présence d'un graphe de compatibilité entre les tâches-couplées impose des contraintes sur celles-ci du point de vue de la possibilité de recouvrir les délais de communication exacte par l'exécution d'une ou plusieurs sous-tâches. Nous présenterons, de nouveau, des résultats de complexité et d'approximation.

La deuxième application concernera l'ordonnancement des tâches viticoles dans un environnement incertain.

Le **chapitre 8** se consacrera à la conclusion de cette thèse et aux perspectives de recherche qui nous semblent pertinentes et envisageables.

CHAPITRE

2 Rappels sur la théorie de la complexité et de l'approximation

2.1 Introduction

Dans la suite de cette thèse, nous focaliserons notre étude sur la classification des problèmes d'ordonnancement selon la théorie de la complexité et de l'approximation. Ce chapitre est centré sur la présentation de quelques notions et résultats en complexité et en approximation. Nous renvoyons à la lecture de certains ouvrages dédiés ([15], [128], [171]) pour des compléments sur les notions introduites dans ce chapitre.

De plus, nous listerons également les problèmes $\mathcal{NP}$-complets utilisés lors des diverses transformations polynomiales que nous avons construites.

2.2 Quelques rappels en théorie de la complexité

2.2.1 Complexité et transformation polynomiale

La classe $\mathcal{P}$ est la classe des problèmes de décision pour laquelle il existe un algorithme (de complexité polynomiale, dépendant de la taille de l'instance) qui permet de répondre par « oui » ou « non » à une instance quelconque. Nous proposons quelques problèmes qui appartiennent à cette classe de complexité :

- le problème de la recherche d'un arbre couvrant de poids de poids k dans un graphe non orienté valué quelconque. Nous pouvons citer les algorithmes de Prim, Kruskal, Sollin et de Boruvka, pour résoudre ce problème.
- le problème de la recherche d'un couplage maximum de poids k dans

un graphe non orienté valué quelconque, en se basant sur l'algorithme de Karp.

– ...

La classe des problèmes $\mathcal{NP}$ est la classe des problèmes pour laquelle nous pouvons vérifier l'existence de la réponse « oui » en temps polynomial. Nous pouvons lister quelques problèmes qui appartiennent à cette classe de complexité :

1. le problème du circuit hamiltonien dans un graphe non orienté quelconque,
2. le problème du stable de taille k,
3. le problème de l'isomorphisme de graphes [1]. Actuellement, nous ne connaissons pas encore l'appartenance de ce problème à la classe $\mathcal{P}$ via un algorithme de complexité polynomiale ou à la classe $\mathcal{NP}-$complet via une transformation polynomiale ou bien à la classe des problèmes intermédiaire.
4. le problème Prime [2]. Il est important de noter, que la classification complète de ce problème a été réalisée par Agrawal et al. dans [5]. Ils ont montré l'appartenance de ce problème à la classe $\mathcal{P}$.

Dans le but de définir la classe des problèmes de décision la plus intéressante, à mon goût, c'est-à-dire la classe des problèmes $\mathcal{NP}$-complet, nous rappelons la définition de la notion de transformation polynomiale. Une transformation polynomiale d'un problème Π_2 à un problème Π_1 (noté par la suite $\Pi_2 \propto \Pi_1$ est une fonction qui associe l'ensemble des instances de Π_2 à l'ensemble à un sous-ensemble des instances de Π_1 tout en satisfaisant les deux conditions suivantes :

1. pour chaque instance I_2 de Π_2, la réponse est « oui » si et seulement si la réponse pour $f(I_2)$ de Π_1 est « oui »,
2. la fonction f est calculable en temps polynomial (dépend seulement de la taille de $|I_2|$).

[1] Nous rappelons qu'en théorie des graphes, un isomorphisme de graphe entre sommets de deux graphes G et H est une bijection $f : V(G) \rightarrow V(H)$ ($V(G)$ désigne l'ensemble des sommets de G) avec la propriété que deux sommets u et v sont adjacents dans G si et seulement si $f(u)$ et $f(v)$ le sont dans H.
Instance : Soient deux graphes G et H
Question : Est-ce que les deux graphes G et H sont isomorphes ?

[2] **Instance :** Soit n un nombre
Question : Est-ce que n est un nombre premier ?

En pratique, la plupart des problèmes d'optimisation combinatoire (du point de vue de la décision) appartiennent à la classe de complexité $\mathcal{NP}-$complet. Cette classe de complexité admet des caractéristiques très fortes. En effet, les problèmes appartenant à cette classe sont les problèmes les plus difficiles de la classe $\mathcal{NP}$, ils sont tous de difficulté équivalente par rapport à leurs résolutions polynomiales et il suffirait de concevoir un algorithme polynomial pour l'un d'entre eux pour que tous les autres fassent appel à lui pour être résolus en temps polynomial.

2.2.2 Le théorème de l'Impossibilité

Il est intéressant de proposer un seuil d'inapproximabilité pour n'importe quel algorithme approché. Pour obtenir ce seuil (ou borne inférieure d'approximation), nous utilisons le résultat de $\mathcal{NP}$-complétude lorsque le critère d'optimalité est égal à une constante.

Théorème 2.2.1 (Théorème de l'impossibilité [50] et [67]) *Étant donné un problème d'optimisation combinatoire* Π *dont la fonction objectif est à valeur entière et soit un entier* c*. Si le problème de décision associé à* Π *et à la valeur* c *est* $\mathcal{NP}$*-complet alors nous ne pouvons pas espérer avoir une heuristique*[3] *pour le problème d'optimisation* Π *ayant un ratio*[4] *inférieur à* $\frac{c+1}{c}$.

Preuve

Nous noterons par $OPT(I)$ la valeur d'une solution optimale pour l'instance I. Si nous supposons qu'il existe un tel algorithme A, alors par contradiction nous montrerons que celui-ci peut-être utilisé pour décider si $OPT(I) \leq c$ en temps polynomial. En effet, nous supposons que l'algorithme A est un algorithme ρ approché pour le problème d'ordonnancement avec $\rho < \frac{c+1}{c}$:

- Si $A(I) < c+1$ alors $OPT(I) \leq c$ et la réponse est oui où $A(I)$ désigne la valeur de l'objectif sur l'instance I à partir de l'algorithme A.
- Si $A(I) \geq c+1$ alors $OPT(I) > c$ car $\frac{A(I)}{c+1} < \frac{OPT(I)}{c}$ (par hypothèse, nous avons un algorithme ρ-approché). La réponse est donc non.

Donc l'algorithme A décide correctement si $OPT(I) \leq c$, et donc $\mathcal{NP} = \mathcal{P}$.

□

[3] Nous appelons heuristique un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable pas nécessairement optimale

[4] La définition de la notion de ratio est précisée dans la Section 2.3.2.

La conséquence de ce théorème est double : il permet d'une part de proposer un seuil d'approximation et d'autre part, il garantit qu'il n'existe pas de schéma d'approximation polynomiale. [5]

2.2.3 Les limites de l'approximation : la technique du « gap »

La *Turing*-réduction (celle que nous utiliserons par la suite (voir Paschos [128]) pour une définition explicite des deux types de réductions classiques Karp et Turing)) divise les valeurs d'un problème d'optimisation en deux sous-ensembles de valeurs distinctes. L'un deux est constitué des instances pour lesquelles la réponse est oui et l'autre pour lesquelles la réponse est non. Nous utiliserons cette technique de séparation des valeurs en deux sous-ensembles pour établir un seuil d'approximation pour n'importe quel algorithme. Ce théorème permet d'utiliser un résultat obtenu sur un problème de décision, pour dériver une borne inférieure d'approximation pour le problème d'optimisation associé au problème de décision.

Théorème 2.2.2 *Soit Π' un problème de décision $\mathcal{NP}$-complet et soit Π un problème d'optimisation dont la fonction objective est la minimisation. Nous supposons qu'il existe deux fonctions polynomiales $f : I_{\Pi'} \rightarrow I_{\Pi}$ (où I_{Π} (resp. $I_{\Pi'}$) désigne l'ensemble des instances de Π (resp. de Π') et $d : I_{\Pi'} \rightarrow \mathbb{N}$ et une constante $gap > 0$ telle que, pour chaque instance x de Π',*

$$S^*(f(x)) = \begin{cases} d(x) \text{ si } x \text{ admet une réponse positive} \\ d(x)(1+gap) \text{ sinon} \end{cases}$$

où S^* désigne une solution optimale.

Alors il n'existe pas d'algorithme r-approché pour le problème Π avec $r < 1 + gap$, sous l'hypothèse que $\mathcal{P} \neq \mathcal{NP}$.

Preuve

Supposons qu'il existe un algorithme r-approché A avec $r < 1+gap$ pour le problème Π. Soit S la solution donnée par l'algorithme A. Cet algorithme pourrait être utilisé pour résoudre le problème Π en un temps polynomial.

Prenons une instance $x \in \Pi'$, nous calculons $f(x)$, et nous appliquons l'algorithme A à $f(x)$. Nous devons distinguer les deux cas suivants :

[5] voir la Section 2.3.3 pour le détail des classes d'approximation

– x est une instance dont la réponse est négative. Par hypothèse, dans ce cas, $S^*(f(x)) = d(x)(1 + gap)$, et donc, $S(f(x), A(f(x)))^6 \geq d(x)(1 + gap)$ avec .
– x est une instance dont la réponse est positive. Dans ce cas sachant que A est un algorithme r-approché, nous avons :

$$\frac{S(f(x), A(f(x)))}{S^*(f(x))} \quad \leq \quad r < 1 + gap$$

Par hypothèse, $S^*(f(x)) = d(x)$, alors $S(f(x), A(f(x))) < d(x)(1 + gap)$.

Ainsi, x est une instance positive de Π' si et seulement si $A(f(x)) < d(x)(1 + gap)$, et A pourrait être utilisé pour résoudre le problème Π' en un temps polynomial. Sachant que Π' est un problème $\mathcal{NP}$-complet, ceci impliquerait que $\mathcal{P} = \mathcal{NP}$. $\square$.

Nous verrons dans la section suivante, que nous pouvons (dans certains cas) déterminer d'une borne supérieure d'approximation pour une heuristique h. ainsi, nous pouvons proposer (si un seuil est possible) un intervalle $[\rho_1, \rho_2]$ (avec ρ_1 (resp. ρ_2) une borne inférieure (resp. supérieure) qui encadre la qualité d'une solution approchée. Le but étant de minimiser l'écart $|\rho_2 - \rho_1|$. A noter que lorsqu'il existe un algorithme approché ayant une performance relative ρ_2 tel que $\rho_1 = \rho_2$, alors nous appellerons cet algorithme un algorithme absolu. Nous pouvons citer deux algorithmes absolus concernant le problème de la coloration des arêtes [7] et le second porte sur le problème du k-centre. [8]

[6] $S(f(x), A(f(x)))$ indique la valeur d'une solution pour une instance $f(x) \in \Pi$ obtenue par l'utilisation de l'algorithme A appliqué à l'instance $f(x) \in \Pi$

[7] **Instance :** Soit un graphe $G = (V, E)$ non orienté.

Question : Est-ce que nous pouvons colorier les arêtes tel que deux arêtes adjacentes admettent des couleurs différentes. Le nombre de couleurs nécessaire à la coloration des arêtes du graphe s'appelle l'indice chromatique.

Il est possible de montrer que décider si un graphe possède un indice chromatique égal à trois est $\mathcal{NP}-$complet (même si le graphe admet un degré au plus trois, voir [90]). L'utilisation du théorème de Vizing (voir [94]) permet de développer un algorithme d'approximation utilisant au plus $k + 1$ couleurs

[8] **Instance :** Soient V un ensemble de n sites, D une matrice où d_{ij} est la distance entre les sites i et j, $k \in \mathbb{N}$ et $\forall i, j, k, d_{ij} \leq d_{ik} + d_{kj}$.

Question : Trouver $S \subseteq V$, avec $|S| = k$, qui minimise $\max_v\{dist(v, S)\}$ où $dist(i, S) = \min_{j \in S}(d_{ij})$.

Il est possible de construire un graphe G', via une transformation polynomiale à partir d'un graphe G du problème ensemble dominant (voir [15]) telle que G' admet un k-centre pour lequel $d_{ij} = 1, \forall\{i, j\} \in E$ si et seulement si G admet un ensemble dominant de taille

2.3 Quelques rappels en théorie de l'approximation

2.3.1 Sur la nécessité de compléter la classification des problèmes

La théorie de la complexité s'intéresse à la classification des problèmes combinatoires selon la difficulté de trouver une solution optimale. Cependant, nous devons compléter et affiner cette classification pour une étude sur la possibilité de trouver et évaluer une solution approchée : c'est la théorie de l'approximation. Affiner la classification est nécessaire et fondamentale pour une bonne compréhension des problèmes d'optimisation combinatoire. En effet, considérons les deux problèmes classiques suivants :

- le problème du stable [9] et,
- le problème de la couverture de sommets [10]

Ces deux problèmes appartiennent à la classe des problèmes $\mathcal{NP}$-difficiles. Cependant, du point de vue de l'approximation ces deux problèmes sont diamétralement opposées. En effet, il est connu que le problème du stable de taille maximum n'est pas approximable, tandis que le problème de la couverture sommet appartient à la classe **APX** (les définitions des classes d'approximation sont données ci-dessous). Ainsi, le comportement concernant la détermination de la qualité de la solution approchée peuvent être très différentes pour deux problèmes $\mathcal{NP}$-complets. Cet exemple est d'autant plus saisissant sachant qu'il existe une transformation polynomiale de l'un à l'autre et réciproquement. Cet exemple illustre bien l'intérêt d'étudier un problème combinatoire du point de vue de la théorie de la complexité et de l'approximation.

2.3.2 Mesure de la qualité d'une solution approchée

Il est nécessaire d'utiliser un critère qui mesure la qualité d'un algorithme approché ou qui donne le seuil d'approximation pour un problème. Nous

k sinon $d \geq 2$. Nous pouvons développer un algorithme 2-approché en se basant sur le graphe carré G^2 dans lequel nous cherchons un ensemble indépendant de taille maximale. (voir [171]).

[9] **Instance :** Un graphe $G = (V, E)$ non orienté

Question : Trouver un sous-ensemble $V' \subseteq V$ tel que $\forall x, y \in V'$, l'arête $[x, y] \notin E$, et V' est maximum

[10] **Instance : Un graphe $G = (V, E)$ non orienté**
Question : Trouver un sous-ensemble $V' \subseteq V$ tel que pour chaque arête $[x, y] \in E$ alors $x \in V'$ ou $y \in V'$, ou $x, y \in V'$

avons choisi la notion de performance relative (ou ratio de performance) associée à une heuristique h noté ρ^h comme critère pour mesurer la qualité. Cette valeur est définie comme étant le maximum sur toutes les instances I entre la valeur d'objectif maximale fournie par l'algorithme h (notée $A^h(I)$) et la valeur d'objectif optimale (notée $OPT(I)$), c'est-à-dire

$$\rho^h = \max_I \frac{A^h(I)}{OPT(I)}.$$

Nous avons de manière évidente que $\rho^h \geq 1$ pour les problèmes de minimisation (resp. $\rho^h \leq 1$ pour les problèmes de maximisation).

La performance relative associée à une heuristique h mesure l'écart maximum entre la valeur donnée par la solution proposée par l'heuristique h, pour une fonction objective donnée, et la valeur donnée par une solution optimale. Les deux principales fonctions objectives à minimiser dans le cadre des problèmes d'ordonnancement sont le temps d'achèvement maximum ou la longueur de l'ordonnancement, et la somme des temps de complétude maximum.

Il existe une autre mesure appelée mesure différentielle [128], nous l'évoquerons dans la conclusion.

2.3.3 Approximation à rapport dépendant de l'instance

Les rapports dépendants de l'instance sont souvent des fonctions soit de la taille même de l'instance d'un problème, soit d'un autre paramètre de cette instance (par exemple, le degré, maximum ou moyen, d'un graphe).

1. **Log-APX**, la classe des problèmes admettant un algorithme approché à rapport classique logarithmique en la taille de l'instance. La couverture d'ensemble appartient à cette classe d'approximation.
2. **Poly-APX**, la classe des problèmes admettant un algorithme approché garantissant un rapport qui est polynomial en la taille de l'instance.
3. **APX**, la classe des problèmes approximables à rapport classique constant, si et seulement s'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 classique de A est borné par r.
4. **PTAS** Un schéma polynomial d'approximation (polynomial-time approximation scheme) pour un problème Π est une famille de d'algorithmes polynomiaux. Un problème Π est dans la classe **PTAS** si et

seulement si il admet un schéma d'approximation classique dont la complexité est polynomiale en la taille de l'instance (mais pouvant être exponentielle en $1/\epsilon$).

5. **FPTAS** Un schéma pleinement polynomial d'approximation (fully polynomial-time approximation scheme) pour un schéma d'approximation de complexité polynomiale à la fois en la taille de l'instance et en $1/\epsilon$. Un problème Π est dans la classe **FPTAS** si et seulement s'il admet un schéma d'approximation classique complètement polynomial.

2.4 Méthodologies

2.4.1 Les méthodes énumératives

Dans cette partie, nous rappellerons les principes de certaines méthodes énumératives les plus classiques. Pour des compléments sur ces méthodes, nous renvoyons le lecteur aux livres suivants [47] et [151].

2.4.1.1 Programmation dynamique

La programmation dynamique est un paradigme de conception introduit dans les années 50 par Bellman. Une solution d'un problème dépend des solutions précédentes obtenues des sous-problèmes. Les sous-problèmes peuvent être en interactions c'est-à-dire un sous-problème peut être utilisé dans la solution de deux sous-problèmes différents.

Si la programmation dynamique est appliquée à un problème combinatoire, alors dans le but de calculer la valeur optimale du critère à optimiser pour n'importe quel sous-ensemble de taille z, nous devons dans un premier temps déterminer la valeur optimale pour chaque sous-ensemble de taille $z-1$. Ainsi, si notre ensemble est caractérisé par un ensemble de n éléments, le nombre de sous-ensembles étudiés est au plus 2^n. Ceci implique que les algorithmes utilisant la programmation dynamique admettent au pire une complexité exponentielle. Cependant, pour les problèmes $\mathcal{NP}$-complets (pas au sens fort [11]), il est possible de construire des algorithmes pseudo-polynomiaux pouvant être résolus de manière efficace avec des instances de tailles raisonnables.

[11] Nous rappelons qu'un problème est dit $\mathcal{NP}$-complet au sens fort, si le problème est $\mathcal{NP}$-complet à cause de sa structure et non pas à cause de sa taille des nombres qui apparaissent dans ses instances, c'est-à-dire, il est $\mathcal{NP}$-complet même si l'on se restreint aux instances où le plus grand nombre est polynomialement borné

Nous pouvons prendre pour illustrer le principe de la programmation dynamique le calcul d'un plus court chemin entre deux sommets dans un graphe valué orienté sans circuit.

Si $\mu = x_0 x_1 \dots x_k$ avec $x_0 = s$ et $x_k = t$ est un plus court chemin de s à t, alors $x_0 x_1 \dots x_{k-1}$ est un plus court chemin de s à x_{k-1}.

$$d^k(i) = \min\{d^{k-1}(i), \min_{v \in \Gamma^-(i)} \{d^{k-1}(v) + w_{vi}\}\}$$

avec $d^k(i)$ désigne la valeur d'un plus court chemin à l'itération k pour le sommet i. Le tableau suivant donne les valeurs $d^k(i)$ pour le graphe de la figure 2.1.

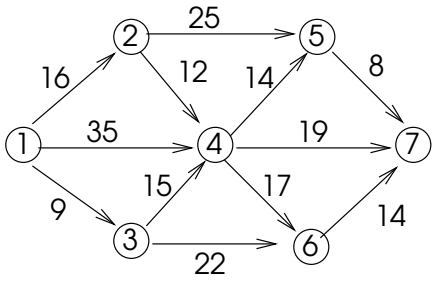


FIG. 2.1 – *Problème d'un plus court chemin*

Itérations\Sommets	1	2	3	4	5	6	7
0	(1,0)	(.,∞)	(.,∞)	(.,∞)	(.,∞)	(.,∞)	(.,∞)
1	(1,0)	(1,16)	(1,9)	(1,35)	(.,∞)	(.,∞)	(.,∞)
2	(1,0)	(1,16)	(1,9)	(3,24)	(2,41)	(3,31)	(4,54)
3	(1,0)	(1,16)	(1,9)	(3,24)	(4,38)	(3,31)	(4,43)
4	"	"	"	"	"	"	"

Pour une colonne z, dans couple (x, y), le paramètre x indique le prédécesseur pour atteindre z, et le paramètre y indique la valeur d'un plus court chemin entre l'origine et le sommet z. Le couple $(., \infty)$ dans une colonne z stipule que le sommet z n'est pas atteignable depuis l'origine. Lorsque deux lignes consécutives sont égales le processus s'arrête. Le principe de la programmation dynamique permet de résoudre via un algorithme pseudo-polynomial le problème d'ordonnancer un ensemble de n tâches indépendantes sur deux processeurs identiques. Ce principe a été utilisé dans [11] pour prouver (dans le modèle à communications hiérarchiques, voir le chapitre 5) l'existence d'un algorithme polynomial pour un problème sur M clusters constitués chacun de m processeurs.

2.4.1.2 La méthode de séparation et d'évaluation

La méthode d'évaluation et de séparation (« Branch and Bound »), est une méthode générique pour résoudre de manière exacte les problèmes d'optimisation combinatoire. C'est une méthode d'énumération implicite : toutes les solutions possibles du problème peuvent être énumérées (la séparation permet d'obtenir une méthode générique pour énumérer toutes les solutions), mais l'analyse des propriétés du problème permet d'éviter l'énumération de larges classes de mauvaises solutions (l'évaluation évite l'énumération systématique de toutes les solutions). Ainsi, dans une bonne application de l'algorithme de séparation et d'évaluation, seules les solutions potentiellement bonnes sont donc énumérées. Nous pouvons citer comme application de cette méthode) ; la résolution du problème d'ordonnancer, sur un processeur unique, une ensemble de tâches indépendantes de durées quelconques soumises à des dates de disponibilité et de fin impérative. Le critère d'optimalité est la minimisation de la longueur de l'ordonnancement (voir [16]). Nous conseillons les références suivantes pour l'application de cette méthode aux problèmes d'ordonnancement ([16], [50], [133]).

2.4.1.3 La programmation par contraintes

Un CSP (Problème de Satisfaction de Contraintes) est un problème modélisé sous la forme d'un ensemble de contraintes posées sur des variables, chacune de ces variables prenant ses valeurs dans un domaine (i.e. un ensemble de valeurs possibles). Le domaine d'une variable peut-être un intervalle de nombres entiers, intervalle d'ensembles par exemple ou continu. Une contrainte implique une ou plusieurs variables, et restreint les valeurs que peuvent prendre simultanément ces variables. Ainsi, trouver une solution à un problème combinatoire modélisé par la programmation par contraintes consiste à décider s'il existe ou non une affectation de toutes les variables telle que l'ensemble des contraintes du problème soient satisfaites. Le principe de la programmation par contraintes est appliqué au problème d'ordonnancement avec contraintes disjonctives qui est une généralisation naturelle des problèmes d'ordonnancement du type job-shop ou open-shop. Nous conseillons la lecture des livres suivants qui traitent de l'utilisation de la programmation par contraintes appliquée à divers problèmes d'optimisations combinatoires et en particulier les problèmes d'ordonnancement [16] et [144].

2.4.1.4 La méthode polyédrale

L'approche polyédrale des problèmes combinatoires est l'une des principales, et satisfaisante approche pour l'analyse, la compréhension et la résolution des problèmes d'optimisation combinatoire. L'étude des enveloppes convexes des vecteurs caractéristiques associés aux solutions réalisables de problèmes particuliers a ouvert la porte à la théorie polyédrale et aux techniques de la programmation linéaire. La plupart des problèmes d'optimisation peuvent s'écrire sous forme d'un programme linéaire en nombres entiers. La méthode classique (en utilisant une approche polyédrale) pour résoudre des problèmes d'optimisation classés $\mathcal{NP}-$difficiles consiste à proposer une « bonne » solution via la relaxation du programme linéaire en nombres entiers modélisant le problème d'optimisation combinatoire. Ensuite, successivement des inégalités sont rajoutées qui sont valides pour toutes les points du polyèdre mais qui ne sont pas satisfaites par la solution optimale courante (donnée par la relaxation du programme linéaire en nombres entiers). A chaque itération, si la solution optimale est réalisable, le problème de séparation (voir [78]) pour la solution optimale courante et l'enveloppe convexe des points réalisables est résolu. Dans le cas contraire, une inégalité violée (un plan sécant), qui est valide pour les points du polyèdre, est déterminée. La méthode de « branch and cut » repose sur l'utilisation des plans sécants.

Concernant l'approche polyédrale, appliquée aux problèmes d'ordonnancement (détermination des enveloppes convexes représentant l'ensemble des contraintes), nous pouvons citer les travaux de Queyranne et Schulz (voir [116], [134], [135], [136], [148], [149]).

2.4.2 Les heuristiques et les algorithmes approchés

Les problèmes d'ordonnancement appartiennent à la classe des problèmes combinatoires. Dans le but de résoudre ces problèmes, il est possible de développer des algorithmes d'optimisation qui détermine une solution optimale. Pourtant, pour la grande majorité des problèmes en ordonnancement, il est impossible de trouver en temps raisonnable une solution optimale. Dans ce cas, il est possible d'utiliser des heuristiques (algorithmes donnant une solution sous-optimale mais ayant une faible complexité).

2.4.2.1 Les algorithmes approchés avec garantie de performance

Les algorithmes approchés sont très populaires, il suffit de consulter la très nombreuse littérature, et reste une branche très active de la recherche en algorithmique. L'intérêt des algorithmes approchés avec garantie de performances réside dans le fait qu'il admettent une solution proposée, par un algorithme polynomial de faible complexité pas trop éloignée (soit dans le sens du pire cas, soit en moyenne) d'une solution optimale. La théorie d'approximation est brièvement présenté au chapitre 2. De nombreux résultats ont été obtenus comme en témoigne la littérature abondante (nous renvoyons aux livres généraux sur l'ordonnancement [16], [50], [133], ...)

2.4.2.2 Les heuristiques de recherche locale

Les méthodes de recherche locales sont très nombreuses, et nous pouvons citer quelques méthodes (les détails et les forces et faiblesses de chaque méthodes ne seront détaillés dans ce manuscrit) :

- le récuit simulé est né d'une analogie entre l'optimisation combinatoire et la thermodynamique ;
- la recherche tabou est une méthode qui consiste à se déplacer de solution en solution en s'interdisant de revenir en une configuration déjà rencontrée.
- le concept des algorithmes génétiques est une heuristique basée sur l'analogie avec le processus d'optimisation de l'évolution de la population d'individus
- les colonies de fourmis, ...
- et bien sûr les méthodes hybrides, c'est-à-dire qui combinent les plusieurs méthodes.

Leurs utilisations pour résoudre des problèmes en ordonnancement couvrent un large spectres de problèmes (Job-shop, problèmes avec contraintes de ressources ...). Nous renvoyons le lecteur à des ouvrages spécialisés sur les méthodes de recherche locale.

2.5 Liste des problèmes $\mathcal{NP}$-complets

Dans cette partie, nous présentons la liste des problèmes $\mathcal{NP}$-complets que nous avons utilisés lors des différentes réductions (voir [67]).

2.5.1 Le problème SAT

Instance du problème SAT :
- Soit $\mathcal{V} = \{x_1, \ldots, x_n\}$ un ensemble de n variables booléennes et $\bar{\mathcal{V}} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ l'ensemble des variables logiques complémentaires avec $x_i \in \mathcal{V}$.
- Soit $\mathcal{C} = \{C_1, \ldots, C_m\}$ une collection de clauses sur $\mathcal{V}$.

Question : Existe-t-il $I : \mathcal{V} \to \{0, 1\}$ une affectation de valeurs de vérité aux variables telle que chaque clause C_i dans $\mathcal{C}$ contienne au moins un littéral mis à vrai par I ? Nous savons que le problème est $\mathcal{NP}$-complet [67].

2.5.2 Le problème $3SAT$

Le problème $3SAT$ est une variante de SAT pour laquelle la longueur de chaque clause est de trois.

2.5.3 Le problème $Monotone$-one-in-$3SAT$

Instance du problème $Monotone$-one-in-$3SAT$:
- Soit $\mathcal{V} = \{x_1, \ldots, x_n\}$ un ensemble de n variables booléennes.
- Soit $\mathcal{C} = \{C_1, \ldots, C_m\}$ une collection de clauses sur $\mathcal{V}$ telle que chaque clause est de taille trois et contient seulement des variables positives

Question : Existe-t-il $I : \mathcal{V} \to \{0, 1\}$ une affectation de valeurs de vérité aux variables telle que chaque clause C_i dans $\mathcal{C}$ contienne exactement un littéral mis à vrai par I ? Nous savons que le problème est $\mathcal{NP}$-complet [67].

2.5.4 Le problème $One - In - (2,3)SAT(2,\bar{1})$

Instances du problème $One - In - (2,3)SAT(2,\bar{1})$ **noté** $\mathcal{P}_1$ **dans la suite :**
- soit $\mathcal{V} = \{x_1, \ldots, x_n\}$ un ensemble de variables logiques et $\bar{\mathcal{V}} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ l'ensemble des variables logiques complémentaires avec $x_i \in \mathcal{V}$.
- soit $\mathcal{C} = \{C_1, \ldots, C_j, C_{j+1}, \ldots, C_q\}$ un ensemble de clauses où $\forall i,\ 1 \leq i \leq j,\ C_i \subset (\mathcal{V} \times \bar{\mathcal{V}})$ et $\forall k,\ (j+1) \leq k \leq q,\ C_k \subset (\mathcal{V})^3$ tel que chaque variable x_i de $\mathcal{V}$ apparaît dans $\mathcal{C}$ deux fois positivement et une fois négativement avec d'une part :

$$\forall x_i \in \mathcal{V}, \begin{cases} occ(x_i\ ;\ C_{k_1}) = 1 \text{ et } occ(x_i\ ;\ C_{k_2}) = 1,\ 1 \leq k_1 \leq j,\ j+1 \leq k_2 \leq q \\ occ(\bar{x}_i\ ;\ C_{k_3}) = 1 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad 1 \leq k_3 \leq j,\ k_1 \neq k_3 \end{cases}$$

et d'autre part, si $x_i \in C_k$ et $\bar{x}_{i'} \in C_k$, $1 \leq k \leq j$, alors $x_i \in C_{k_m}$ et $x_{i'} \in C_{k_l}$, avec $k_m \neq k_l$ et $j+1 \leq k_m$, $k_l \leq q$.
La fonction $occ(x, C)$ désigne le nombre d'occurrences de la variable x dans la clause C.

Question : Existe-t-il $I : \mathcal{V} \to \{0,1\}$ une affectation de valeurs de vérité aux variables telle que chaque clause C_i contienne exactement un littéral mis à vrai par I ? Par abus de langage, une telle affectation sera dite satisfaisante pour la donnée du problème $\mathcal{P}_1$.

Exemple : pour illustrer une instance du problème $One\text{-}In\text{-}(2,3)SAT(2,\bar{1})$, nous considérons la formule logique suivante : $(\bar{x}_0 \vee x_3) \wedge (\bar{x}_3 \vee x_0) \wedge (\bar{x}_4 \vee x_2) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_5 \vee x_1) \wedge (\bar{x}_2 \vee x_5) \wedge (x_0 \vee x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5)$. Pour cette instance, la réponse ici est « oui », car il suffit que les variables x_0 et x_3 s'évaluent à « vrai » et les autres à « faux ». L'instance précédente proposée est la plus petite instance possible en terme de taille.

2.5.5 Le problème de la clique maximum

Instance : Soit $G = (V, E)$ un graphe, soit k un entier

Question : Est-ce qu'il existe un sous-graphe $G' = (V', E')$ dans G avec G' est une clique et $|V'| = k$?

2.5.6 Le problème d'un sous-ensemble indépendant équilibré dans un graphe biparti

Instance : Soit un graphe $B = (X \bigcup Y, E)$ avec $|X| = |Y| = n$ non orienté, biparti et équilibré et un entier k.

Question :

Est-ce-qu'il y a dans le graphe B, un ensemble de $2k$ sommets indépendants dont k appartiennent à X et k à Y ?

2.5.7 Le problème 3-partition

Instance : Un ensemble $Q = \{e_1, \ldots, e_{3q}\}$ avec :
$\sum_{e_i \in Q} e_i = qE$ et $E/4 < e_i < E/2$ pour tout i.

Question : Peut-on partitionner Q en q sous-ensembles $Q_1, \ldots, Q_q$ avec :
$\sum_{e_i \in Q_j} e_i = E$ et $E/4 < e_i < E/2$ pour tout j ?

Si on a une solution à 3-Partition, alors on a, du fait de l'inégalité $E/4 < e_i < E/2$, $|Q_j| = 3$ pour tout j.

Par la suite, on pourra supposer que $e_i \leq \cdots \leq e_{3q}$.

2.5.8 Le problème partition en triangles

Instance : Soit $G = (V, E)$ un graphe, avec $|V| = 3q, q \in \mathbb{N}$.

Question : Les sommets de G peuvent ils être partionnés en q ensembles $V_1, V_2, \ldots, V_q$, 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 $\{u_i, v_i\}, \{v_i, w_i\}$ et $\{u_i, w_i\}$ appartiennent à E ?

CHAPITRE

3 Définition, analyse et classification des problèmes d'ordonnancement

Dans ce chapitre, nous allons introduire les notions basiques utilisées dans la théorie de l'ordonnancement statique. Nous commencerons par la notion de tâches, d'environnement de processeurs et les critères d'optimalité. Nous poursuivrons par la notion de granularité, de graphe de précédence valué. Nous préciserons l'approche qui nous a guidé pour résoudre les divers problèmes d'ordonnancement. Nous finirons par donner la notation à trois champs $(\alpha|\beta|\gamma)$, dont le premier concerne l'environnement des processeurs, le second les caractéristiques des tâches et le troisième le critère d'optimalité.

Ce chapitre est très largement inspiré par [16].

3.1 Définition des problèmes d'ordonnancement

3.1.1 Description des tâches

Dans la littérature, en général (nous adopterons cette définition dans la suite), les problèmes d'ordonnancement se déclinent de la manière suivante[1] : nous disposons de deux ensembles $\mathcal{T} = \{T_1, \ldots, T_n\}$ de n tâches, $\mathcal{P} = \{\pi_1, \ldots, \pi_m\}$ de m processeurs. L'ordonnancement consiste à affecter l'ensemble $\mathcal{T}$ des tâches, soumises à des contraintes imposées, à l'ensemble des processeurs $\mathcal{P}$. Il existe en général deux contraintes en théorie de l'ordonnancement :

[1] Nous avons omis l'ensemble des ressources à l'exécution d'une tâche pour ne pas alourdir d'une part la présentation, et d'autre part dans le reste de ce manuscrit nous considérons des modèles pour lesquels l'utilisation de ressources supplémentaires n'est pas pris en compte.

- chaque processeur est capable d'exécuter au plus une tâche à chaque instant,
- chaque tâche ne peut-être exécutée qu'au plus par un seul processeur[2].

3.1.2 Environnement des processeurs

Maintenant caractérisons les processeurs. Ils peuvent être soit parallèles, exécutant les mêmes fonctions, soit dédiés i.e. spécialisés pour l'exécution de certaines tâches. Les trois types de processeurs de machines parallèles se distinguent par leurs vitesses de traitement des tâches. Si tous les processeurs de l'ensemble $\mathcal{P}$ admettent la même vitesse de traitement, nous dirons que les processeurs sont identiques. Si les vitesses de traitements des processeurs diffèrent, mais chaque vitesse b_i (facteur d'accélération du traitement de la tâche) de chaque processeur est constant et ne dépend pas de la tâche à exécuter, alors nous dirons que les processeurs sont uniformes. En dernier lieu, si les vitesses des processeurs dépendent des tâches à ordonnancer, nous dirons que les processeurs de la machine parallèle sont généraux.

Dans le cas où le les processeurs sont dédiés, nous pouvons présenter trois modèles : `flow shop, open shop` et `job shop`. Nous renvoyons les personnes intéressées pour ces trois modèles à la lecture des livres [16], [99], [133]. Ces trois modèles n'étant pas étudiés dans cette thèse, ils ne seront pas décrits.

En général, une tâche $T_j \in \mathcal{T}$ est caractérisée par les données suivantes :

1. un vecteur de temps d'exécution $p_j = [p_{1j}, p_{2j}, \ldots, p_{mj}]^T$, où p_{ij} désigne le temps nécessaire au processeur π_i d'exécuter T_j. Dans le cas où tous les processeurs sont identiques, nous avons $p_{ij} = p_j, i = 1, \ldots, m$. Si les processeurs de $\mathcal{P}$ sont uniformes, nous avons $p_{ij} = p_j/b_i, i = 1, \ldots, m$ où p_j représente la vitesse d'exécution standard et b_i facteur d'accélération du traitement de la tâche par le processeur P_i.
2. une date de disponibilité, notée r_j, qui désigne la date à partir de laquelle la tâche T_j est prête à être exécutée. Si les dates de disponibilités sont équivalentes pour toutes les tâches de $\mathcal{T}$, alors nous pouvons supposer que $r_j = 0, \forall j$.
3. une date d'échéance d_j, qui spécifie la date limite (souhaité) à laquelle la tâche T_j devra être exécutée.

[2]Dans certains modèles hiérarchiques du type tâches modelables ou malléables [39, 57], une tâche peut s'exécuter sur plusieurs processeurs (les communications sont intégrées dans la durée d'exécution des tâches). Cependant la durée dépend du nombre de processeurs exécutant cette tâche.

4. une date de fin impérative $\tilde{d}_j$, date à laquelle la tâche T_j devra avoir fini son exécution.
5. un poids (priorité) w_j, qui représente l'urgence relative de la tâche T_j.

Dans toute la suite, nous supposerons que les paramètres liés aux tâches p_j, $r_j, \tilde{d}_j, d_j$ et w_j sont des entiers.

Un ordonnancement est dit préemptif si chaque tâche peut-être préemptée (interrompue) à n'importe quel moment et son exécution peut-être reprise également à n'importe quel moment. Si la préemption des tâches est interdite, nous dirons que l'ordonnancement est non-préemptif.

Les tâches de $\mathcal{T}$ à exécuter sur les processeurs de $\mathcal{P}$ peuvent être dupliquées. En effet, dans le but de réduire l'influence des délais de communication (voir ci-après), dans le cas où la duplication des tâches est autorisée, des copies de tâches peuvent être produites. Nous supposerons par la suite, que les tâches originelles et leurs copies admettent une même date de début d'exécution.

Sur l'ensemble des tâches $\mathcal{T}$, des contraintes de précédence peut-être définies. $T_i \prec T_j$ signifie que la tâche T_j ne pourra commencer son exécution que si la tâche T_i a fini la sienne. En d'autres termes, nous pouvons définir sur l'ensemble $\mathcal{T}$ une relation de précédence donnée par $\prec$. Les tâches de l'ensemble $\mathcal{T}$ sont dites dépendantes si au moins deux tâches de $\mathcal{T}$ admettent un relation de précédence. Dans le cas contraire, nous dirons que les tâches sont indépendantes. Les tâches soumises à des relations de précédence, seront représentés par un graphe orienté (valué ou non, selon la présence ou non de délais de communication) où les sommets correspondent aux tâches et les arcs les contraintes de précédence. Nous supposerons que nous n'avons pas d'arc transitif dans le graphe de précédence.

Une tâche T_j est dite disponible à l'instant t si $r_j \leq t$ et si tous ces prédécesseurs (en respectant les contraintes de précédence) ont fini leur exécution avant l'instant t.

3.1.3 Critères d'optimalité

Maintenant, nous donnons les définitions concernant les ordonnancements et les critères d'optimalité. Un ordonnancement est une affectation des tâches de $\mathcal{T}$ sur les processeurs de $\mathcal{P}$ telles que les conditions suivantes soient satisfaites :

- à chaque instant un processeur exécute au plus une tâche et chaque tâche est exécutée par un seul processeur,
- chaque tâche T_j est exécutée durant l'intervalle $[r_j, \infty)$.

- toutes les tâches sont ordonnancées.
- si deux tâches T_i et T_j admettent une relation de précédence $T_i \prec T_j$, alors la tâche T_j ne peut commencer son exécution qu'après la fin d'exécution de T_i.
- dans le cas d'ordonnancements non préemptifs, aucune tâche ne peut-être interrompue. Dans le cas contraire, le nombre de préemptions est fini.
- dans le cas où la duplication est non autorisée, aucune tâche ne peut-être dupliquée. Dans le cas contraire, nous supposerons que le nombre de copies d'une tâche est fini.

Pour représenter les ordonnancements, nous utiliserons un diagramme de Gantt. Un exemple d'utilisation du diagramme de Gantt est donné à la figure 3.2. Dans cette figure, nous avons à notre disposition trois processeurs identiques sur lesquels nous devons exécuter un ensemble de huit tâches, ayant des durées différentes, soumises à des contraintes de précédence données par la graphe de précédence de la figure 3.1. Dans cette figure les valuations sur un sommet x correspondent à son temps d'exécution sur un processeur. Pour chaque tâche $T_j, i = 1, \dots n$ exécutée dans un ordonnancement donné, nous pouvons calculer la valeur de chaque paramètre suivant :

- le temps de complétion noté $C_j = t_j + p_j$ où t_j désigne la date de début d'exécution de la tâche T_j,
- le temps d'attente $F_j = C_j - (r_j + p_j)$,
- le retard algébrique noté $L_j = C_j - d_j$,
- le retard absolue $D_j = \max\{C_j - d_j, 0\}$,
- l'avance de la tâche $E_j = \max\{d_j - C_j, 0\}$,

Pour illustrer ces notions, nous utiliserons l'ordonnancement donné par la figure 3.2. Nous pouvons facilement calculer le vecteur C, nous obtenons $C = [3, 4, 5, 6, 1, 8, 8, 8, 8]$. De plus, si nous ajoutons le vecteur suivant pour les dates échues $d = [5, 4, 5, 3, 7, 6, 9, 12]$, nous pouvons calculer les vecteurs L, D et E. Ainsi, nous obtenons $L = [-2, 0, 0, 3, -6, 2, -1, -4]$, $D = [0, 0, 0, 3, 0, 2, 0, 0]$ et $E = [2, 0, 0, 0, 6, 0, 1, 4]$.

Pour évaluer les ordonnancements, nous pouvons utiliser les trois principaux critères d'optimalité :

- la longueur de l'ordonnancement défini par $C_{max} = \max_j\{C_j\}$,
- la moyenne $\bar{F} = \frac{1}{n}\sum_{j=1}^{n} F_j$, ou dans le cas pondéré $\bar{F_w} = \sum_{j=1}^{n} w_j F_j / \sum_{j=1}^{n} w_j$,
- le maximum des retards algébriques $L_{max} = \max_j\{L_j\}$,

Pour certaines application, d'autres critères d'optimalité sont étudiés :

- $\bar{D} = \frac{1}{n}\sum_{j=1}^{n} D_j$, ou la version pondérée $\bar{D_w} = \sum_{j=1}^{n} w_j D_j / \sum_{j=1}^{n} w_j$,

– $\bar{E} = \frac{1}{n}\sum_{j=1}^{n} E_j$, ou la version pondérée $\bar{E}_w = \sum_{j=1}^{n} w_j E_j / \sum_{j=1}^{n} w_j$,
– le nombre de tâches en retard $U = \sum_{j=1}^{n} U_j$ avec $U_j = 1$ si $C_j > d_j$, et 0 sinon, ou la version pondérée $U_w = \sum_{j=1}^{n} w_j U_j$.

Ainsi, nous pouvons calculer les valeurs de ces critères d'optimalité pour les tâches soumises aux contraintes de précédence donnée par la figure 3.1. Nous obtenons $C_{max} = 8$, $\bar{F} = 43/8$, $L_{max} = 3$, $\bar{D} = 5/8$, $\bar{E} = 13/8$ et $U = 2$. Les versions pondérées peuvent être évaluées en spécifiant le vecteur w.

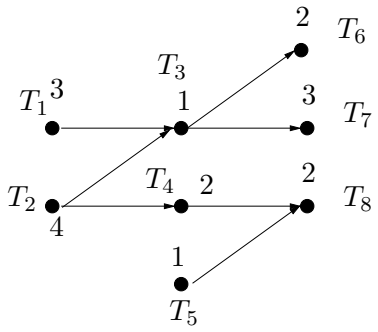


FIG. 3.1 – *Un graphe de précédence*

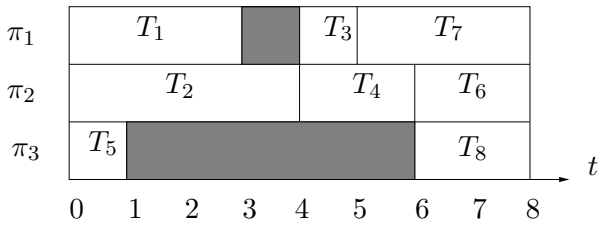


FIG. 3.2 – *Un ordonnancement pour le graphe de précédence donné par la figure 3.1*

3.1.4 Notion de granularité

Une caractéristique importante d'un graphe de précédence avec délais de communications est la taille du *grain* choisi pour la découpe en tâches. La notion de granularité cherche à représenter le rapport entre la durée des calculs effectués pour l'exécution d'une tâche et les délais de communication. Dans le cadre des modèles idéalisés où les communications sont considérées comme instantanées (par exemple le modèle PRAM), la mesure cruciale de la

complexité parallèle est la profondeur du graphe de précédence. Or il s'avère que dans la pratique, on doit tenir compte du délai de communication entre l'instant où une information est produite par un processeur, et l'instant à partir duquel cette information peut être utilisée par un autre processeur. Ceci induit un surcoût lié aux communications qui dépend, entre autres, de la quantité d'information échangée. L'ordonnancement des tâches de l'application va dépendre dans ce cas non seulement des durées d'exécution des tâches mais aussi des temps de communications entre celles-ci.

Définition 3.1.1 *Nous appelons la granularité d'un graphe $G = (V, E)$ le rapport entre le plus grand temps de communication sur le plus petit temps de calcul d'une tâche :*

$$\Phi = \frac{\max_{(i,j)\in E} c_{ij},}{\min_{i\in V} p_i}$$

Dans le cas où $\Phi \geq 1$ (resp. $\Phi \leq 1$) alors nous qualifierons le problème d'ordonnancement comme étant soumis aux grands délais de communications (resp. aux petits délais de communications). Dans le cas où l'on considère les petits délais de communications, la structure des ordonnancements est simplifiée ce qui permet de réduire la combinatoire des solutions possibles car une synchronisation s'effectue entre la date de début d'exécution des tâches et les communications. En effet, en présence des grands délais de communications, une communication entre deux tâches T_i et T_j peut-être recouverte par l'exécution d'une ou plusieurs tâches. Cette caractéristique rend les problèmes avec grands délais de communications beaucoup plus difficiles (voir le chapitre 4 pour un rappel des résultats pour le modèle homogène). Un cas particulier intéressant apparaît quand $\forall i \in v,\ p_i = 1$ et $\forall (i,j) \in E,\ c_{ij} = 1$ c'est-à-dire le problème $UET - UCT$.

3.2 Approches et stratégies de résolution des problèmes d'ordonnancement

Dans cette partie, nous présentons plusieurs voies et stratégies pour résoudre un problème d'ordonnancement. Ces stratégies sont également valables pour tous types de problèmes d'optimisation combinatoire.

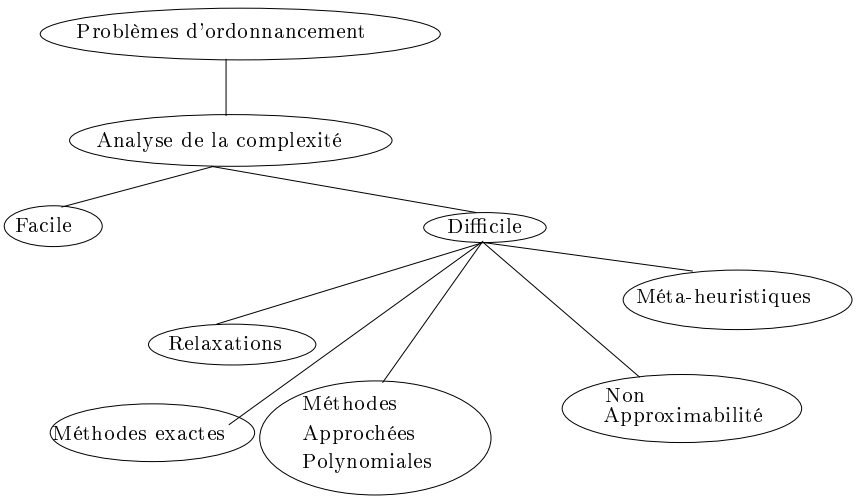


FIG. 3.3 – *Approche synthétique face à un problème d'optimisation combinatoire et en particulier les problèmes d'ordonnancement*

Nous nous sommes focalisés sur la classification au sens de la théorie de la complexité et de l'approximation des problèmes d'ordonnancement. Il est important de noter que le fait de classifier un problème d'ordonnancement comme $\mathcal{NP}$-complet n'est qu'une première étape d'analyse de la résolution d'un problème et non la dernière (dans le cas contraire, nous essayons de développer des algorithmes exacts de plus faible complexité). Après une telle preuve nous allons nous limiter à :

- des versions plus simples du problème. Nous pouvons relaxer les hypothèses du problème originel et résoudre le problème relaxé. La solution du problème relaxé peut servir de base pour l'étude du problème originel. Dans le cas des problèmes d'ordonnancement les relaxations peuvent se porter :
  - sur la possibilité d'autoriser la préemption des tâches, même si dans le problème originel le caractère préemptif des tâches n'était pas autorisé,
  - de manière similaire, l'autorisation de dupliquer des tâches,
  - sur la restriction à des durées d'exécution, et/ou de délai de communication, en prenant des durées unitaires quand dans le problème originel les durées sont arbitraires,
  - sur la structure du graphe de précédence, en considérant des graphes ayant des spécifications fortes (arbre, graphe biparti, ...),
  - sur l'environnement des machines, nous pouvons relaxer la contrainte

sur le nombre limité de machines en considérant un nombre de machines arbitrairement grand,
- sur la fonction objective.

La relaxation de ces hypothèses du problème originel permet de simplifier la combinatoire des problèmes. Par exemple, dans le cas *UET-UCT* (Unit Execution Time Unit Communication Time, temps d'exécution et de communication unitaires), les dates de début d'exécution des tâches et les délais de communication entre deux tâches adjacentes dans le graphe de précédence sont synchronisées. Ainsi, les dates de début d'exécution des tâches sont entières et il ne peut y avoir de conflit concernant le placement des tâches sur les processeurs pour obtenir un ordonnancement réalisable. En effet, entre deux instants consécutifs entiers (t et $t+1$) les processeurs sont soient inactifs ou soient ils exécutent une tâche, mais pas les deux.

Pour ces problèmes fortement contraints, nous essayons de déterminer la frontière entre l'existence d'un algorithme polynomial et la $\mathcal{NP}$-complétude. Par exemple, il a été montré que dans le cas où la duplication des tâches est autorisée, sur une infinité de processeurs, le problème *UET-UCT* est polynomial [49]. A contrario, pour le même problème sans la duplication, nous savons qu'il n'existe pas de $\mathcal{PTAS}$ (voir [172]).

- à la recherche des algorithmes polynomiaux avec garantie de performances non triviales [3].

  Ces algorithmes sont basés :
  - sur la construction d'une liste de priorité sur les tâches (c'est l'algorithme générique en ordonnancement),
  - sur la relaxation des contraintes d'intégrité d'un programme linéaire en nombres entiers, avec une phase d'arrondis. Initialement, le problème d'ordonnancement est formulé par le programme linéaire en nombres entiers. Dans certains cas, il n'est possible de formuler le problème d'ordonnancement que par un programme linéaire où la valeur des variables est dans un intervalle $I \subseteq \mathbb{R}$ (au lieu d'être dans un sous-ensemble $S \subseteq \mathbb{N}$ dans le cas d'un programme linéaire en nombres entiers). Dans ce cas là, nous n'avons simplement qu'à

[3] Nous étudierons dans le reste de ce manuscrit que des algorithmes approchés avec une étude dans le pire des cas. Nous discuterons dans le conclusion la possibilité de compléter cette classification en procédant à une autre étude.Nous entendons par algorithme polynomial avec garantie de performance triviale, la pire exécution d'un algorithme de liste (pour ce type d'algorithme une liste de priorité est créée, et à chaque instant pour chaque processeur inactif une tâche disponible est exécutée sur le-dit processeur.

prendre en compte la phase d'arrondis. De plus, sachant que la solution du relaxé est une borne inférieure de toute solution optimale, nous pouvons conclure aisément.

- à la recherche d'un seuil à partir duquel trouver un algorithme approché garantissant une performance meilleure devient impossible (sauf si $\mathcal{P} = \mathcal{NP}$). Ceci va nous donner les limites sur les garanties de performance que l'on peut obtenir par des algorithmes polynomiaux pour un problème d'ordonnancement donné. La technique utilisée pour un résultat de non-approximabilité (ou seuil d'approximation) consiste à démontrer un résultat de $\mathcal{NP}$-complétude pour le problème de décision et d'invoquer le théorème de l'impossibilité [50], qui permet de relier un résultat de $\mathcal{NP}$-complétude pour un problème de décision au problème d'optimisation associé. Une nouvelle approche consiste à utiliser la « technique du gap » (voir le chapitre 2 pour la présentation de la méthode) pour obtenir un résultat de non-approximabilité pour une certaine fonction objectif f à partir d'un résultat de $\mathcal{NP}$-complétude obtenu pour une autre fonction objectif f'.

Quand toutes les tentatives d'obtenir des résultats analytiques semblent épuisés, nous pouvons orienter la recherche de résultats vers des méthodes exactes (très gourmandes en temps et en puissance de calcul) ou vers une approche stochastique.

- Les méthodes exactes (programmation dynamique, méthode par séparation et évaluation) ou des méta-heuristique (recherche tabou, recuit simulé, colonies de fourmis, ...) peuvent être des alternatives pour étudier les problèmes.
- La seconde approche consiste à effectuer une analyse stochastique des données en procédant à des hypothèses sur les tâches (fréquence, périodicité ...) (voir les références [109],[50] pour l'étude des problèmes d'ordonnancement cyclique).

La discussion précédente est résumée par le diagramme donné par la figure 3.3.

3.3 Classification des problèmes d'ordonnancement déterministes

3.3.1 Présentation de la notation à trois champs

La 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. [77] et par Błażewicz et al. [19] et nous proposons une extension.

Pour définir un problème d'ordonnancement, il est commode d'introduire trois paramètres α, β, γ. 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 $\alpha = \alpha_1(\alpha_2)$[4] définit l'environnement des processeurs (nombre de processeurs, modèle hiérarchique, topologie, ...)
  $*\alpha_1 = \in \{P, \bar{P}, P_m, \varnothing\}$ et $\alpha_2 \in \{P, \bar{P}, P_m, Q, R, O, F, J\}$
  - Si $\alpha_1 \neq \varnothing$ alors nous sommes en présence d'un modèle hiérarchique.
  - Si $\alpha_i = P$ signifie que les processeurs sont identiques et que leur nombre m est une entrée du problème.
  - Si $\alpha_i = \bar{P}$ ou $\alpha_2 = \infty$ signifie que les processeurs sont identiques, et que leur nombre est suffisant ou non bornée.
  - Si $\alpha_i = P_m,\ i \in \{1, 2\}$ signifie que le nombre de processeurs identiques est fixé.
  - Si $\alpha_i = Q$, les processeurs sont uniformes,
  - Si $\alpha_i = R$, les processeurs sont généraux,
  - Si $\alpha_i = O$, les processeurs sont dédiés (modèle Open-Shop),
  - Si $\alpha_i = F$, les processeurs sont dédiés (modèle Flow-Shop),
  - Si $\alpha_i = J$, les processeurs sont dédiés (modèle Job-shop).
- Le paramètre β définit le type d'application et ces caractéristiques c'est-à-dire le graphe de précédence, le coût des communications, les temps d'exécution des tâches, l'autorisation ou pas de la duplication, l'autorisation ou pas de la préemption.
  De manière synthétique, on obtient $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5$ où :
  $*\beta_1 \in \{prec, arbre, chaine, \ldots, .\}$
  - Si $\beta_1 =$prec (le graphe est quelconque).
  - Si $\beta_1 =$arbre (le graphe est un arbre)
  - $\vdots$
  - Si $\beta_1 = .$ (les tâches sont indépendantes)

  $*\beta_2 \in \{com, c_{jk}, c, .\}$
  - Si $\beta_2 =$com (les temps de communication sont donnés par le graphe).
  - Si $\beta_2 = c_{jk}$ (représente le temps de communication entre la tâche j et la tâche k).
  - Si $\beta_2 = c$ (le temps de communication est constant et égal entre chaque paire de tâches).
  - $\beta_2 = .$ (il n'existe pas de communication).

[4] Il se peut que des combinaisons n'aient pas de sens.

$*\beta_3 \in \{p_j, .\}$
- Si $\beta_3 = p_j = 1$ (toutes les tâches ont une durée unitaire).
- Si $\beta_3 = .$ (les durées d'exécution sont définies par le graphe).

$*\beta_4 \in \{dup, .\}$
- Si $\beta_4 = dup$ (on autorise la duplication des tâches).
- Si $\beta_4 = .$ (on n'autorise pas la duplication des tâches).

$*\beta_5 \in \{pmtp, .\}$
- Si $\beta_5 = pmtp$ (on autorise l'interruption d'une tâche).
- Si $\beta_5 = .$ (on n'autorise pas l'interruption d'une tâche).

- Le paramètre γ définit la fonction objective ou le critère d'optimalité. Dans toute la suite de cette thèse, nous considérons les deux fonctions objectives suivantes :
  - la minimisation de la longueur de l'ordonnancement, notée $C_{max} = \max_i\{t_i + p_i\}$ (où p_j est la durée d'exécution de la tâche j et t_j est la date de début d'exécution) par la suite, qui est le temps d'achèvement. De plus, C^{opt}_{max} désignera la longueur optimale de l'ordonnancement.
  - la minimisation de la somme des temps de complétude noté $\sum_j C_j$ avec $C_j = t_j + p_j$.

Exemple :

1. Le problème $P||C_{max}$ signifie que nous devons ordonnancer un ensemble de tâches indépendantes non-préemptibles, de durées quelconques, sur un ensemble de processeurs identiques et dont le critère est la minimisation de la longueur de l'ordonnancement.
2. Le problème $\bar{P}|prec; p_1 = 1; c_{ij} = 1, dup|\sum_j C_j$ signifie que nous devons ordonnancer un ensemble de tâches, pour lesquelles la duplication est autorisée, de durées unitaires, soumises à des contraintes de précédence quelconques sur un ensemble de processeurs de taille non bornée. De plus, il existe une communication potentielle, unitaire entre les tâches i et j, et le but est minimiser la somme des temps de complétude des tâches.

3.3.2 Commentaires sur cette notation et dépendances

Avec cette notation à trois champs, nous voyons bien les possibilités qui nous sont offertes sur la multiplicité des problèmes que nous pouvons étudier :

- Sur les fonctions objectives (le champ γ) : nous avons focalisé notre étude sur deux critères (C_{max} et $\sum_j C_j$). D'autres critères sont possibles (somme des temps de complétude pondérés $\sum_j w_j C_j$, ... (voir

[133])). De plus, nous pouvons étudier les divers critères d'optimisation soit séparément(problème mono-critère) soit de manière simultanément (ce sont les problèmes multi-critères [166]).
Enfin, nous pouvons considérer des problèmes de maximisation ou de minimisation.

- De même, le champs β étant constitué de multiples sous-champs, il est possible d'imposer des contraintes fortes sur certains sous-champs (autorisation de la duplication, possibilité d'avoir des tâches préemptives, ...). Ces restrictions permettent d'avoir une influence sur la classe de complexité du problème étudié.
- Avec l'apparition des nouvelles architectures, le champ α peut-être un critère d'étude : architecture reconfigurable, prise en compte de la topologie des processeurs, regroupement de processeurs sous forme de clusters, vitesses de processeurs différentes ...

La figure 3.4 décrit quelques transformations polynomiales basiques entre différents problèmes d'ordonnancement. Pour chaque graphe de dépendance, les problèmes présentés différent seulement d'un paramètre (par exemple, par type de graphe de précédence, sur le graphe d) de la figure 3.4, les flèches indique les directions des transformations polynomiales.

- a) l'environnement des processeurs,
- b) la possibilité ou non de la préemption,
- c) la possibilité ou non de la duplication,
- d) les contraintes de précédence,
- e) la disponibilité des tâches,
- f) la durée d'exécution des tâches,
- g) les critères d'optimalité

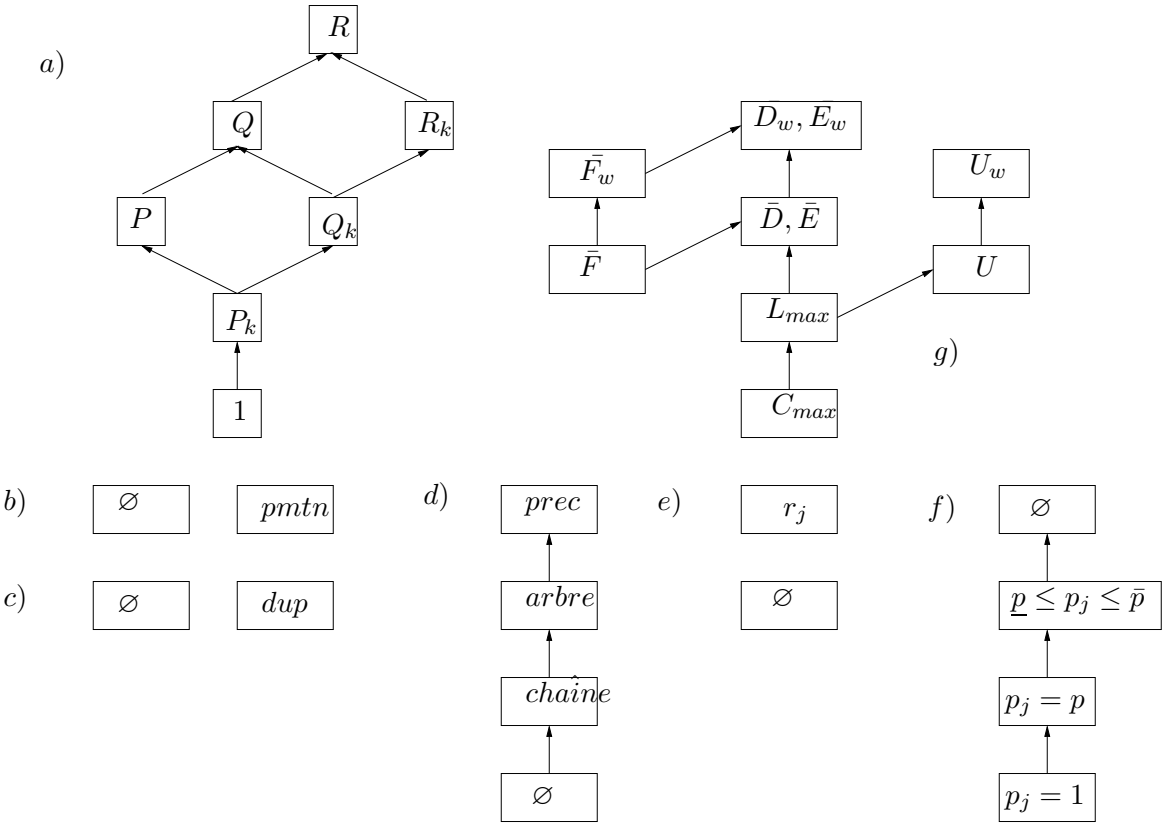


Fig. 3.4 – *Illustrations des différentes relations entre des paramètres particuliers*

Deuxième partie

Ordonnancement pour le parallélisme

CHAPITRE

4 Communications homogènes

4.1 Présentation et motivations

4.1.1 Introduction

Avant toute présentation des résultats sur le modèle homogène, je souhaiterai commencer par une remarque historique sur les algorithmes approchés pour les problèmes d'ordonnancement.

Remarque : Il est important de noter qu'à notre connaissance, Coffman *et al.* [53] ont établi le premier résultat d'approximation (avec garantie de performance non triviale [1]) pour un problème $\mathcal{NP}$-difficile (problème qui consiste à ordonnancer un ensemble de tâches indépendantes de durée quelconque sur un nombre limité de processeurs identiques $P||C_{max}$). L'algorithme polynomial proposé est basé sur la construction d'une liste de priorité, définie à partir de la durée d'exécution des tâches, et l'affectation des tâches sur les processeurs suit le principe suivant : dès qu'un processeur π est inactif, la première tâche i disponible dans la liste de priorité sera exécutée sur le processeur π. Cet algorithme est l'algorithme fondamental en ordonnancement.

Le modèle d'ordonnancement avec `communications homogènes` a été très largement étudié dans les années 90 du point de vue de la complexité et de l'approximation (par la communauté internationale et plus particulièrement française) comme en témoigne la très nombreuse littérature portant sur ce sujet (nous pouvons citer ces quelques références [16], [48], [50]). Le modèle a été introduit dans le but de remédier à la faiblesse du modèle d'ordonnancement sans communication introduit dans les années $60-70$ et également répondre à l'évolution des machines parallèles. Le modèle sans délai de communication est caractérisé par la définition suivante :

[1]

Soit $G = (V, E)$ un graphe de précédence, $\forall (i, j) \in E$, c'est-à-dire que i précède j dans le graphe de précédence, noté $i \prec j$, nous avons $t_i + p_i \leq t_j$. Remarquons que l'ensemble des contraintes de précédence peut-être vide ($E = \varnothing$), alors conformément au chapitre 3, les tâches sont dîtes indépendantes.

Il s'avère que dans la pratique, on doit tenir compte du délai de communication entre l'instant où une information est produite par un processeur, et l'instant à partir duquel cette information peut-être utilisée par un autre processeur. Ce modèle a été introduit dans un premier temps par Rayward-Smith [141] dans un soucis de développer des modèles plus réalistes. Ainsi, dans ce modèle la notion de délai de communication entre deux tâches i et j soumises à une contrainte de précédence est prise en compte. Ceci induit un surcoût lié aux communications qui dépend, entre autres, de la quantité d'information échangée. L'ordonnancement des tâches de l'application va dépendre dans ce cas non seulement des durées d'exécution des tâches mais aussi des temps de communications potentielles entre celles-ci.

Soit $G = (V, E)$ un graphe de précédence (avec V l'ensemble des tâches à exécuter et E représentant les contraintes de précédence), nous associons une durée d'exécution p_i pour chaque tâche i et entre tout couple de tâches i et j, soumises à une contrainte de précédence, nous associons une valeur c_{ij} représentant la communication potentielle entre la tâche i et la tâche j si ces deux tâches s'exécutent sur des processeurs différents. C'est le *modèle d'ordonnancement à communications homogènes*.

Formellement, dans le *modèle d'ordonnancement à communications homogènes* nous associons une valeur $c_{ij}, \forall (i, j) \in E$ telle que :

- si $\pi_i = \pi_j$ alors $t_i + p_i \leq t_j$
- sinon si $\pi_i \neq \pi_j$ alors $t_i + p_i + c_{ij} \leq t_j$

où t_i désigne le début d'exécution de la tâche i et π_i le processeur sur lequel i s'exécute.

Dans la suite, nous supposerons que $\forall i \in V, p_i = 1$ et $\forall (i, j) \in E, c_{ij} = c$ (problème UET-LCT, (Unit-Execution-Time Large-Communication-Time, voir la section 4.2)).

4.1.2 Analyse des résultats existants

La figure 4.1 récapitule les principaux résultats de complexité dans le modèle à *communications homogènes*, dont le critère est la minimisation de longueur de l'ordonnancement, tandis que la figure 4.2 récapitule ceux concernant l'approximation. A partir de ces arbres de classification, nous pouvons

noter qu'il existe pas encore de résultats en approximation et en complexité pour le problème d'ordonnancer des tâches unitaires sur un nombre illimité de processeurs en présence des grands délais de communication. Nous allons combler ce vide (voir la section 4.2.5). De plus, nous pouvons noter l'influence de l'autorisation ou non de la duplication (introduite par Papadimitriou et Yannakakis [126] afin de réduire l'influence des délais de communication sur la durée de l'ordonnancement) sur la recherche d'une solution optimale.

En effet, le résultat classique Hoogeveen et al. [89] indique qu'il n'existe pas d'algorithme d'approximation avec une performance relative à moins de 7/6 pour le problème UET-UCT. Tandis que si la duplication est autorisée Colin et Chrétienne [49] ont proposé un algorithme polynomial de complexité quadratique mais avec beaucoup de duplicats.

Pour finir, nous pouvons également noter les limites de l'introduction de la duplication. Lorsque le nombre de processeurs est limité ou lorsque les grands délais de communications sont considérés ; l'autorisation ou non de la duplication n'a aucune influence sur la complexité de la recherche d'une solution optimale. Tous les problèmes sont $\mathcal{NP}$-complets.

Voici les explications des acronymes :

- $UET - UCT$: Unit-Execution-Time Unit-Communication-Time, i.e $c_{ij} = p_i = 1$;
- LCT : Large-Communication-Time, $\min c_{ij} \geq \max p_i$
- *dup* : la duplication des tâches est autorisée
- Sans com. : absence de communication

4.1.3 Présentation du chapitre

Le chapitre se décompose de la manière suivante :

- dans la section 4.2, nous procéderons à un rappel en complexité et en approximation sur le problème avec des grands temps de communication et nous établirons la $\mathcal{NP}$-complétude du problème pour la minimisation de la longueur de l'ordonnancement dans la section 4.2.3.
- dans la section 4.2.4, nous proposerons un algorithme polynomial pour $C_{max} = c + 2$ avec $c \in \{2, 3\}$.
- dans la section 4.2.5 nous développerons un algorithme ρ-approché avec $\rho = \frac{2(c+1)}{3}$ (meilleure borne connue à notre connaissance).
- dans la section 4.3 nous effectuerons une analyse des résultats obtenus.

Ces travaux ont été mené avec J.C. König, J. Palaysi et F. Moulaï (voir [70], [73] et [74]).

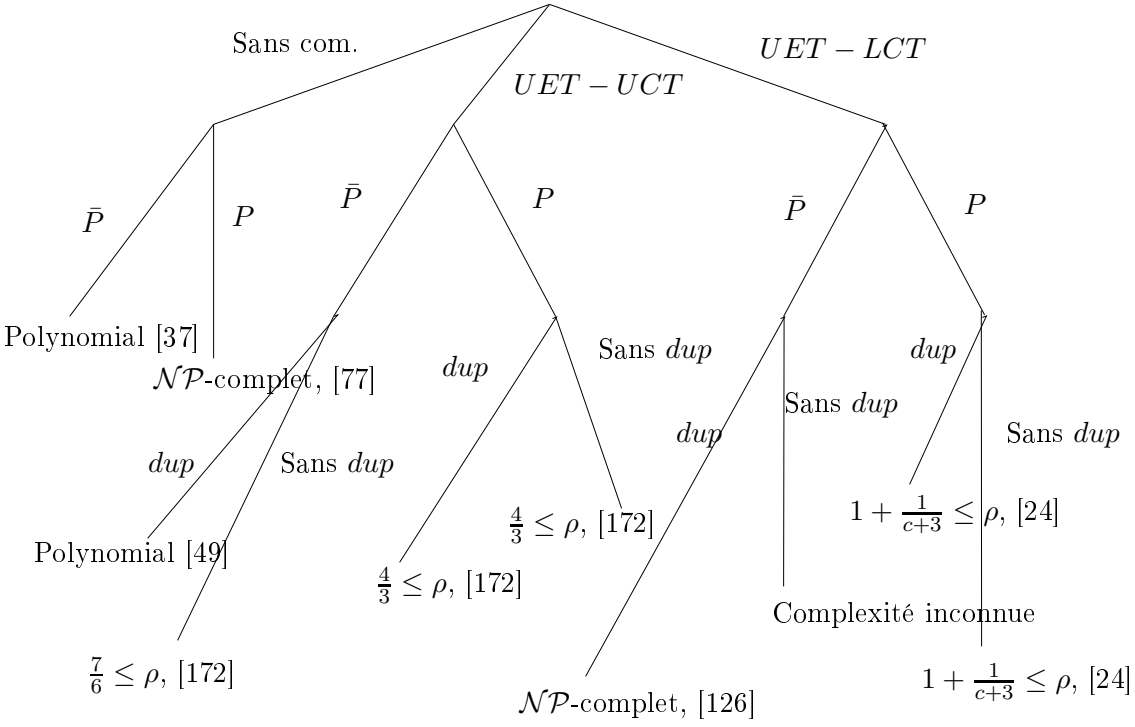


Fig. 4.1 – *Principaux résultats de complexité pour le modèle UET pour la minimisation de la longueur de l'ordonnancement*

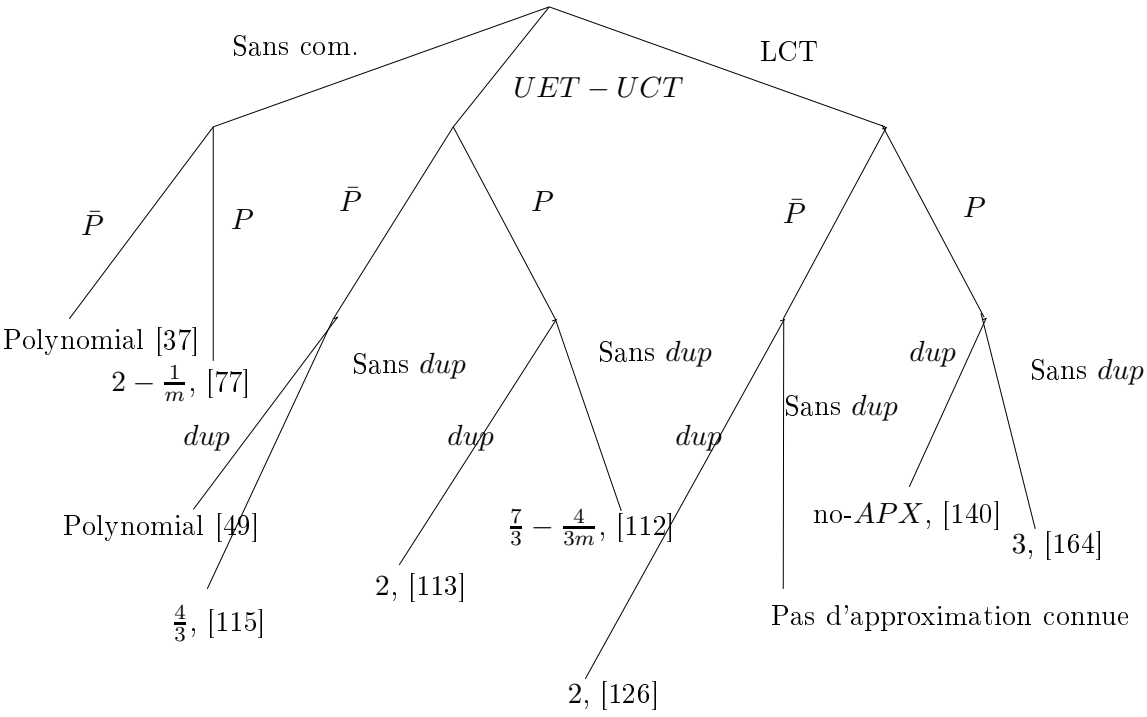


Fig. 4.2 – *Principaux résultats d'approximation pour le modèle UET pour la minimisation de la longueur de l'ordonnancement*

4.2 Les grands délais de communication

4.2.1 Rappel des résultats existants sur les grands temps de communication

Si nous considérons le problème d'un graphe de précédence avec des grands délais de communications, dont les tâches admettent des durées exécution unitaire $UET - LCT$, sur un nombre limité de machine, Bampis *et al.* in [24] ont prouvé que le problème de décision noté par $P|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ pour $C_{\max} = c + 3$ est un problème $\mathcal{NP}$-complet, et pour $C_{\max} = c + 2$ (pour le cas $c = 2$), ils développent un algorithme polynomial. Cet algorithme ne peut-être étendu au cas $c \geq 3$. Leur preuve est basée sur une réduction à partir du problème $\mathcal{NP}$-complet *Balanced Bipartite Complete Graph*, *BBCG* [67, 145]. Ainsi, Bampis *et*

Machines	c_{ij}	C_{max}	Complexité	Bornes inférieures	Références
$\bar{P}$	$c=1$	5	Polynomial		[172]
$\bar{P}$	$c=1$	6	$\mathcal{NP}$-complet	$7/6 \leq \rho$	[172]
P	$c=1$	3	Polynomial		[131]
P	$c=1$	4	$\mathcal{NP}$-complet	$5/4 \leq \rho$	[172]
$\bar{P}$	$c \geq 2$	$> c$	?	?	?
P	$c \geq 2$	$c+1$	Polynomial		[24]
P	$c \geq 2$	$c+3$	$\mathcal{NP}$-complet	$1+1/(c+3) \leq \rho$	[24]

TAB. 4.1 – Résultats de complexité pour le modèle à communication homogène en présence ou non de grands délais de communication

al. [24] démontrent que le problème $P|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ admet un seuil d'approximation de valeur $(1 + \frac{1}{c+3})$.

Le tableau 4.1 rappelle les principaux résultats de complexité et de non-approximation. Il est important de noter que dans le cas de l'existence d'une borne pour tout algorithme d'approximation nous garantit que nous n'avons pas de schéma d'approximation polynomial. Dans le cas d'un graphe de précédence quelconque Rapine [140] propose une borne inférieure en $O(c)$ pour un algorithme de liste en présence des grands délais de communication (dans le cas où la duplication est autorisée il démontre que tout algorithme de liste possède une garantie $\theta(\sqrt{c})$, ce qui montre le gain mais aussi la limitation de cette approche). Les résultats d'approximation sont donnés par le tableau 4.2.

Il est important de noter, nous n'avons pas à notre connaissance trouvé de rapport d'approximation un algorithme ayant un ratio strictement inférieur à $c+1$. Cet algorithme consiste à exécuter les tâches sans prédécesseur, à procéder à une phase de communication, à exécuter les tâches dont les prédécesseurs ont été ordonnancés, à procéder à une nouvelle phase de communication et ainsi de suite ...

Machines	c_{ij}	Approximation	Références
$\bar{P}$	$c = 1$	$\rho \leq 4/3$	[112]
P	$c = 1$	$\rho \leq 7/3$	[111]
$\bar{P}$	$c \geq 2$	2 pour l'arbre	[110]
$\bar{P}$	$c \geq 2$	?	?
P	$c \geq 2$	$O(c)$	[140]

TAB. 4.2 – Résultats d'approximation pour le modèle à communication homogène en présence ou non de grands délais de communication

4.2.2 Motivation

Nous considérons le problème de la minimisation de la longueur de l'ordonnancement. Au vue des résultats, il est légitime de se poser la question suivante : « **Est-ce que l'écart sur les seuils d'approximation pour le problème $UET-LCT$ est identique à celui du problème $UET-UCT$ en environnement de processeurs différents (un nombre infini et fini de processeurs » ?** Dans la suite, nous allons répondre à cette question.

4.2.3 Seuil d'approximation pour les grands délais de communication

Le challenge pour le problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ est de déterminer une borne inférieure (ou seuil d'approximation pour tout algorithme d'approximation). Ainsi dans cette partie, nous allons dans un premier temps prouver que le problème $\bar{P}|prec; c_{ij} = c \geq 3; p_i = 1|C_{max}$ n'admet pas d'algorithme approché avec un ratio plus petit que $(1 + \frac{1}{c+4})$ pour la minimisation de la longueur de l'ordonnancement (resp. $1 + \frac{1}{2c+5}$, pour la minimisation de la somme des temps de complétude). Nous montrons également, dans le cas où les délais de communications sont égaux à deux ($c = 2$), que le problème devient $\mathcal{NP}-$complet pour $C_{max} = 6$.

4.2.3.1 Minimisation de la longueur de l'ordonnancement

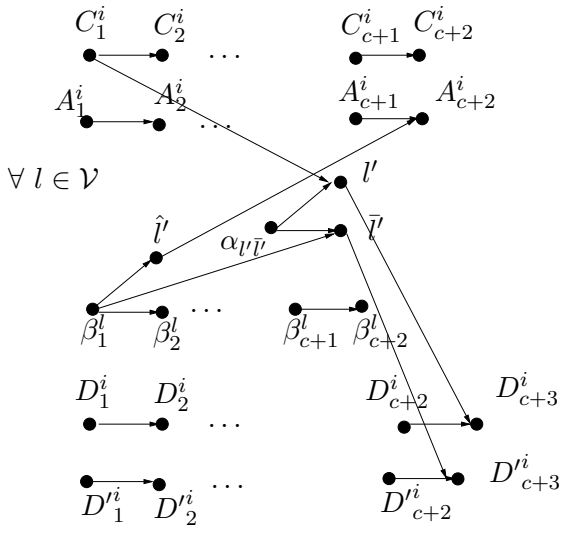


FIG. 4.3 – *Graphe partiel du graphe de précédence utilisé pour la preuve de la $\mathcal{NP}$-complétude du problème d'ordonnancement $\bar{P}|prec; c_{ij} = c \geq 3; p_i = 1|C_{max}$.*

Théorème 4.2.1 *Le problème si une instance de $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max}$ admet un ordonnancement de longueur au plus $(c+4)$ est $\mathcal{NP}$-complet avec $c \geq 3$.*

Preuve

Il est facile de voir que $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max} = c + 4 \in \mathcal{NP}$.

Notre preuve est basée sur une réduction à partir du problème $\mathcal{P}_1$ (voir le chapitre 2 section 2.5.4 pour la description de ce problème). Soit π^* une instance de $\mathcal{P}_1$, nous construisons une instance π du problème $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max} = c + 4$, de la manière suivante :

n désigne le nombre de variables de π^*.

1. $\forall l \in \mathcal{V}$, nous introduisons $(c + 6)$ tâches-variables : $\alpha_{l'\bar{l}'}$, l', $\bar{l}'$, $\hat{l}'$, β_j^l avec $j \in \{1, 2, \ldots, c + 2\}$. Nous ajoutons les contraintes de précédence suivantes : $\alpha_{l'\bar{l}'} \rightarrow l'$, $\alpha_{l'\bar{l}'} \rightarrow \bar{l}'$, $\beta_1^l \rightarrow \hat{l}'$, $\beta_1^l \rightarrow \bar{l}'$, $\beta_j^l \rightarrow \beta_{j+1}^l$ avec $j \in \{1, 2, \ldots, c + 1\}$.
2. Pour chaque clause de longueur trois notée par $C_i = (y \vee z \vee t)$, nous introduisons $2 \times (2 + c)$ tâches-clauses C_j^i et A_j^i, $j \in \{1, 2, \ldots c + 2\}$, avec les contraintes de précédence : $C_j^i \rightarrow C_{j+1}^i$ et $A_j^i \rightarrow A_{j+1}^i$, $j \in$

$\{1, 2, \ldots, c+1\}$. Nous ajoutons les contraintes de précédence suivantes : $C_1^i \rightarrow l$ avec $l \in \{y', z', t'\}$ et $l \rightarrow A_{c+2}^i$ avec $l \in \{\hat{y}', \hat{z}', \hat{t}'\}$.

3. Pour chaque clause de longueur deux notée par $C_i = (x \vee \bar{y})$, nous introduisons $(c+3)$ tâches-clauses D_j^i, $j \in \{1, 2, \ldots, c+3\}$ avec pour contraintes de précédence : $D_j^i \rightarrow D_{j+1}^i$ avec $j \in \{1, 2, \ldots, c+2\}$ et $l' \rightarrow D_{c+3}^i$ avec $l \in \{x, \bar{y}\}$.

La construction proposée ci-dessus est illustrée par la figure 4.3. Cette réduction est clairement calculable en un temps polynomial.

Remarque : $\bar{l}'$ est élément de la clause C' de longueur deux associé au chemin $D'^i_1 \rightarrow D'^i_2 \rightarrow \ldots D'^i_{c+2} \rightarrow D'^i_{c+3}$

- Supposons qu'il existe un ordonnancement de longueur au plus $(c+4)$. Nous allons montrer qu'il existe une affectation $I : \mathcal{V} \rightarrow \{0, 1\}$ des valeurs de vérité aux variables satisfaisant l'instance π^* du problème $\mathcal{P}_1$.
  Dans un premier temps nous pouvons remarquer que si $c \geq 3$ alors $2c+2 > c+4$ et, chaque chemin A_j^i, β_j^x, C_j^i ou $D_{j'}^i$ avec $j \in \{1, 2, \ldots, c+2\}$ et $j' \in \{1, 2, \ldots, c+3\}$ doit être exécuté sur le même processeur. Ainsi, deux chemins ne peuvent être exécutés sur le même processeur.
  Notation : Dans la suite nous noterons par P_A (resp. P_C) l'ensemble des $\frac{n}{3}$ processeurs qui exécutent un chemin A_j^i (resp. un chemin C_j^i). Notons que par définition du problème $\mathcal{P}_1$, n'importe quelle instance admet $\frac{n}{3}$ clauses de longueur trois où n correspond au nombre de variables. De la même manière, nous notons par P_β (resp. P_D) l'ensemble des n processeurs qui exécutent un chemin β_j^x (resp. un chemin D_j^i).

Lemme 4.2.1 *Pour $C_{max} = c + 4$: la décision d'affecter la valeur « vrai » à la variable x si et seulement si la tâche-variable x' est exécutée sur le processeur du chemin P_C produit une solution réalisable.*

Preuve

Dans le but de respecter un ordonnancement réalisable de longueur $(c+4)$, dans un premier temps, nous pouvons produire à partir d'une transformation polynomiale, le début d'exécution des tâches-variables l', $\bar{l}'$ et $\hat{l}'$, et que les processeurs sur lequel ces tâches doivent être exécutées, sont donnés par les remarques suivantes :
$\forall l \in \mathcal{V}$:

– chaque tâche-variable l' est ordonnancée sur le processeur que P_C à l'instant 3 ou sur un autre processeur on a processeur de P_D à l'instant $(c+2)$ ou $(c+3)$,

- chaque tâche-variable $\bar{l}'$ est exécutée sur un processeur de P_β au slot 3 ou sur un processeur de P_D au slot $(c+2)$ or $(c+3)$,
- chaque tâche-variable $\hat{l}'$ est exécutée sur un processeur de P_β au slot 2 ou 3 ou sur un processeur de P_A au slot $(c+2)$ ou $(c+3)$,
- les tâches-variables $\bar{l}'$ et $\hat{l}'$ ne peuvent être exécutées ensemble sur un processeur de P_β (ils admettent un prédécesseur commun).

Notations et propriétés : pour chaque $l \in \mathcal{V}$, nous pouvons associer les trois tâches l', $\bar{l}'$, $\hat{l}'$. Nous notons par $X = \{l'|l \in \mathcal{V}\}$, $\bar{X} = \{\bar{l}'|l \in \mathcal{V}\}$ et $\hat{X} = \{\hat{l}'|l \in \mathcal{V}\}$ trois ensembles de tâches. Pour chaque sous-ensemble A de $\bar{X}$ (resp. $\hat{X}$), nous pouvons associer un sous-ensemble B de X de la manière suivante : $l' \in B$ *si et seulement si* $\bar{l}' \in A$ (resp. $\hat{l}' \in A$).
Considérons les ensembles suivants :

- $X_1 = \{l' \backslash \pi(l') \in P_C\}$ où $\pi(l')$ désigne le processeur sur lequel la tâche l' est exécutée,
- $X_2 = \{l' \backslash \pi(l') \in P_D\}$,
- $X_3 = \{l' \backslash \pi(\bar{l}') \in P_\beta\}$,
- $X_4 = \{l' \backslash \pi(\bar{l}') \in P_D\}$,
- $X_5 = \{l' \backslash \pi(\hat{l}') \in P_\beta\}$,
- $X_6 = \{l' \backslash \pi(\hat{l}') \in P_A\}$.

Considérons que $\mathcal{X}_i = |X_i|$ pour $i \in \{1, \ldots, 6\}$.
Nous pouvons dériver à partir de la construction de l'instance du problème d'ordonnancement la tableau suivant :

	P_C	P_β	P_A	P_D
x'	X_1			X_2
$\bar{x}'$		X_3		X_4
$\hat{x}'$		X_5	X_6	

A partir, du tableau précédent, en utilisant la variable $\mathcal{X}_i$, nous obtenons le système d'inéquations suivant :

$$\mathcal{X}_1 + \mathcal{X}_2 = n \tag{4.1}$$
$$\mathcal{X}_3 + \mathcal{X}_4 = n \tag{4.2}$$
$$\mathcal{X}_5 + \mathcal{X}_6 = n \tag{4.3}$$
$$\mathcal{X}_1 \leq \frac{n}{3} \tag{4.4}$$
$$\mathcal{X}_6 \leq \frac{2n}{3} \tag{4.5}$$
$$\mathcal{X}_3 + \mathcal{X}_5 \leq n \tag{4.6}$$
$$\mathcal{X}_2 + \mathcal{X}_4 \leq n \tag{4.7}$$

Nous précisons en détail de ces équations :

- Pour les équations (4.1), (4.2) et (4.3) toutes les tâches de l'ensemble X, $\bar{X}$ et $\hat{X}$ doivent être exécutées.
- Pour chaque équation (4.4), sur le processeur qui exécute le chemin C_j^i de la clause $C_i = (y \vee z \vee t)$, une tâche-variable au plus parmi les trois tâches-variables y', z', t' peut être exécutée. En effet, toutes les tâches-variables l' admettent un successeur qui sera exécuté sur le processeur que P_D. Si elle est allouée sur le même processeur qui exécute les tâches du chemin P_C, elle ne peut être affectée avant le slot 3, et donc la tâche-variable $\alpha_{l'\bar{l}'}$ peut être affectée sur le même processeur qui devient saturé. Alors, nous obtenons $|X_3| < |P_C|$.
- Pour chaque équation (4.5), précise que chaque processeur exécutant les tâches d'un chemin P_A admet deux instants d'inactivités et $|P_A| = \frac{n}{3}$.
- Pour l'équation (4.6), toutes les tâches-variables $\bar{l}'$ ou $\hat{l}'$ qui sont exécutées sur le processeur du chemin P_β doivent être terminées avant le slot 3 (elles admettent un successeur exécuté sur un autre processeur). Alors la tâche-variable $\alpha_{l'\bar{l}'}$ doit être ordonnancée sur le même processeur qui devient saturé. Donc, seulement une des tâches-variables $\bar{l}'$ et $\hat{l}'$ peut être exécutée sur le processeur du chemin P_β et donc, $|X_3| + |X_5| \leq |P_\beta|$.
- Pour l'équation (4.7), il est clair que $|P_D| = n$ et il existe un instant d'inactivité au plus, sur chaque processeur de P_D.

Dans un premier temps, nous avons $\mathcal{X}_3 + \mathcal{X}_5 = n$ (en effet, nous avons $\mathcal{X}_3 + \mathcal{X}_4 + \mathcal{X}_5 + \mathcal{X}_6 = 2n$ et $\mathcal{X}_6 \leq \frac{2n}{3}$, $\mathcal{X}_4 \leq \frac{n}{3}$, alors $\mathcal{X}_3 + \mathcal{X}_5 \geq n$).
Dans un second temps, $\forall l'$ seulement une des tâches-variables $\bar{l}'$ et $\hat{l}'$ peut-être affectée sur le processeur P_β, ainsi nous obtenons $X_3 \cap X_5 = \emptyset$. Par conséquent, nous avons $X_3 \cup X_5 = X$. Comme l'ensemble X_4 (resp. X_6) est le complémentaire de l'ensemble X_3 (resp. X_5) nous avons $X_4 \cup X_6 = X$. De plus, si la tâche-variable l' est ordonnancée sur le processeur de P_C alors la tâche-variable $\alpha_{l'\bar{l}'}$ est affectée sur le même processeur. Ainsi la tâche-variable $\bar{x}'$ ne peut être exécutée avant l'instant $(c+2)$, et donc est ordonnancée sur le processeur de P_D. Nous pouvons déduire que $X_1 = X_4$ (les deux ensembles admettent la même cardinalité).
Finalement, nous avons $X_1 \cup X_2 = X$, $X_3 \cup X_4 = X$, $X_5 \cup X_6 = X$, $X_4 \cup X_6 = X$, $X_3 \cup X_5 = X$, $X_1 = X_4$ et donc $X_1 = X_4 = X_5$ et $X_2 = X_3 = X_6$.

Nous pouvons déduire des équations précédentes

$$\mathcal{X}_1 = \mathcal{X}_4 = \mathcal{X}_5 = \frac{n}{3}$$

et

$$\mathcal{X}_2 = \mathcal{X}_3 = \mathcal{X}_6 = \frac{2n}{3}.$$

Alors, si nous affectons la valeur « vrai » à la variable l si et seulement si la tâche-variable l' est exécutée sur le processeur de P_C. Il est clair que la clause de longueur trois nous avons un et un seul littéral à « vrai ».

Considérons $C = (x \vee \bar{y})$, une clause de longueur 2.

- Si $x' \in X_1 \implies y' \in X_4 \implies y' \in X_1$. La première implication (resp. la seconde) est due au fait que chaque processeur exécutant un chemin P_D est saturé ($x_2 + x_4 = n$) (resp. $X_1 = X_4$). Seul le littéral x est « vrai » entre les variables x et $\bar{y}$.
- Si $x' \in X_2 \implies y' \in X_3 \implies y' \in X_2$. La première (resp. la seconde) implication est due au fait qu'il existe un seul instant inactivité sur chaque processeur exécutant le chemin P_D (resp. $X_3 = X_2$). Seulement le littéral $\bar{y}$ est « vrai » entre les variables x et $\bar{y}$.

En conclusion, il existe un seul littéral à « vrai » par clause. *Ceci conclut la preuve du Lemme 4.2.1.*

□

- Réciproquement, supposons qu'il existe une affectation $I : \mathcal{V} \to \{0, 1\}$ des valeurs de vérité aux variables satisfaisant l'ensemble des clauses. Alors un ordonnancement de longueur au plus $(c+4)$ peut être obtenu de la manière suivante.

  Supposons le littéral mis à vrai dans clause $C_i = (y \vee z \vee t)$ est t. Donc, la tâche-variable t' (resp. y' etz') est affectée au slot 2 (resp. au slot $(c+2)$) sur le même processeur que le chemin P_{C_i} (resp. les chemins P_D et $P_{D'}$, où D et D' indique que les clauses de longueur deux où les variables y et z apparaissent). Les $\frac{2n}{3}$ autres tâches-variables y' qui ne sont pas encore ordonnancées, sont affectées au slot 3 sur le processeur P_β comme la tâche-variable $\alpha_{y'\bar{y}'}$. La tâche-variable $\hat{t}'$ (resp. $\hat{y}'$ et $\hat{z}'$) est exécutée à l'instant $t = 2$ (resp. $(c+2)$ et $(c+3)$) sur le processeur du chemin P_β (resp. P_A).

  Ceci conclut la preuve du Théorème 4.2.1.

□

4.2.3.2 Étude du cas $c = 2$

Dans cette partie, nous allons considérer le cas spécial où le délai de communication potentiel entre chaque paire de tâches i et j est égal à $c = 2$:

$\forall (i,j) \in E$,

- si $\pi^i = \pi^j$ alors $t_j \geq t_i + p_i$
- sinon $t_j \geq t_i + p_i + 2$.

Nous étudions la complexité de ce problème, et nous allons prouver la $\mathcal{NP}$-complétude de $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max}$ pour $C_{max} = 6$. Nous rappelons que le problème $UET - UCT$ ($\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$) est $\mathcal{NP}$-complet (resp. polynomial) pour $C_{max} = 6$ (resp. $C_{max} = 5$), (voir [131],[172]).

La preuve de la $\mathcal{NP}$-complétude pour le problème $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max}$ est basée sur une réduction effectuée à partir du problème $\mathcal{NP}$-complet $\mathcal{P}_1$ différente de celle présentée ci-dessus en appliquant $c = 2$.

Théorème 4.2.2 *Le problème de décider si une instance de $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max}$ admet un ordonnancement de longueur au plus six est $\mathcal{NP}$-complet.*

Preuve

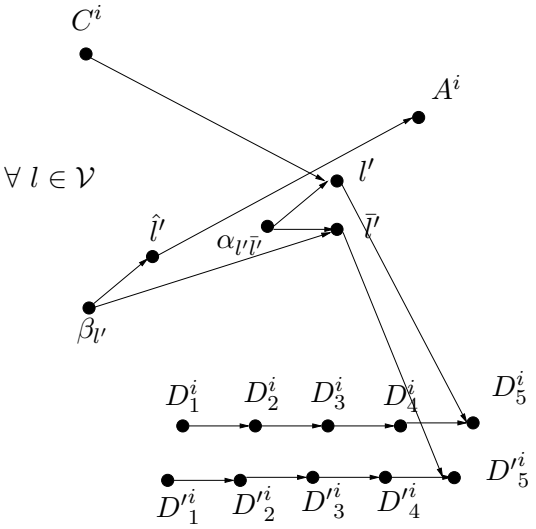


Fig. 4.4 – *Graphe partiel du graphe de précédence pour la démonstration de la $\mathcal{NP}$-complétude pour le problème d'ordonnancement $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max}$.*

Avant de proposer une transformation polynomiale, il est important de noter que la transformation suggérée dans la preuve du Théorème 4.2.1 ne peut être utilisée pour la démonstration de la $\mathcal{NP}$-complétude du problème d'ordonnancement pour le cas particulier $c = 2$.

Nous supposons que nous appliquons la transformation polynomiale avec $c = 2$. Considérons $C_{j_1} = (y \vee z \vee t)$ une clause de longueur trois. Nous supposerons que la tâche-variable y est exécutée à l'instant $t = 2$ sur le processeur π. Pour toutes les clauses de longueur trois, par construction, un chemin de longueur quatre est créé (ici, le chemin $C_1^{j_1} \rightarrow C_2^{j_1} \rightarrow C_3^{j_1} \rightarrow C_4^{j_1}$). La tâche-variable y associée à la tâche y admet deux prédécesseurs. Nous supposons que les tâches-clauses $C_1^{j_1}$ sont exécutées à $t = 0$. Alors, la tâche-variable $\alpha_{yy'}$ est ordonnancée à $t = 1$. Alors, les tâches-clauses restantes du chemin $C_2^{j_1} \rightarrow C_3^{j_1} \rightarrow C_4^{j_1}$ peuvent être exécutées sur un autre processeur que π. Sur le processeur π, les tâches-variables z et $\bar{x}$ et la tâche-clause C_{j_2} où $C_{j_2} = (z \vee \bar{x})$ sont ordonnancées. En répétant cette affectation pour les clauses de longueur trois, le nombre de tâches qui sont exécutées sur P_C est n (rappelons que P_C est l'ensemble des $\frac{n}{3}$ processeurs qui exécutent un chemin C_j^i avec $j \in \{1, \ldots, 4\}$ pour une clause de longueur trois C_i). Dans la preuve précédente, nous certifions que pour la construction suggérée, le nombre de tâches devant être ordonnancées est $n/3$. Contradiction.

Revenons à la démonstration du Théorème :

Il est facile de voir que $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max} = 6 \in \mathcal{NP}$.

Notre preuve est basée sur une réduction à partir de $\mathcal{P}_1$.

Soit une instance π^* de $\mathcal{P}_1$, nous construisons une instance π du problème $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max} = 6$, de la manière suivante :

1. $\forall l \in \mathcal{V}$, nous introduisons 5 tâches-variables : $\alpha_{l'}$, l', $\bar{l}'$, $\hat{l}'$, $\beta_{l'}$. Nous ajoutons les contraintes de précédence : $\alpha_{l'} \rightarrow l'$, $\alpha_{l'} \rightarrow \bar{l}'$, $\beta_{l'} \rightarrow \hat{l}'$, $\beta_{l'} \rightarrow \bar{l}'$.
2. Pour chaque clause de longueur trois notée $C_i = (y \vee z \vee t)$, nous introduisons les tâches-clauses C^i etA^i. Nous ajoutons comme contraintes de précédence : $C^i \rightarrow l$ avec $l \in \{y', z', t'\}$ et $l \rightarrow A^i$ avec $l \in \{\hat{y}', \hat{z}', \hat{t}'\}$.
3. Pour chaque clause de longueur deux noté $C_i = (x \vee \bar{y})$, nous introduisons cinq tâches-clauses D_j^i, $j \in \{1, 2, \ldots, 5\}$ avec pour contrainte de précédence : $D_j^i \rightarrow D_{j+1}^i$ avec $j \in \{1, 2, 3, 4\}$ et $l \rightarrow D_5^i$ avec $l \in \{x', \bar{y}'\}$.

La construction ci-avant est illustrée par la figure 4.4. Clairement la transformation est calculable en temps polynomial.

La preuve complète et détaillée est donnée dans [74].

□

Nous pouvons déduire des deux Théorèmes précédents, le corollaire classique sur le seuil d'approximation :

Corollaire 4.2.1 *Il n'existe pas d'algorithme d'approximation pour le problème* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ *avec une performance relative bornée par* $1 + \frac{1}{c+4}$ *sous l'hypothèse que* $\mathcal{P} \neq \mathcal{NP}$.

Preuve

La preuve du Corollaire 4.2.1 est une conséquence immédiate du Théorème de l'Impossibilité, (voir [50], [67], section 2.2.2).

□

4.2.3.3 Minimisation de la somme des temps de complétude

Dans cette partie, nous montrerons qu'il n'existe pas d'algorithme approché pour le problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|\sum_j C_j$ avec une performance inférieure à $1 + \frac{1}{2c+5}$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$. Ce résultat est obtenu par une transformation polynomiale issue de la preuve du Théorème 4.2.2 et de la technique du gap présentée dans la chapitre 2 (voir également [88]).

Théorème 4.2.3 *Il n'existe pas d'algorithme approché pour le problème* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|\sum_j C_j$ *avec pour performance bornée par* $1 + \frac{1}{2c+5}$ *sous l'hypothèse* $\mathcal{P} \neq \mathcal{NP}$.

Preuve

Nous supposons qu'il existe un tel algorithme noté par A avec pour performance garantie de $1 + \frac{1}{2c+5}$. Soit I une instance du problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ obtenue par réduction (voir Théorème 4.2.2).

Soit I' une instance du problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|\sum_j C_j$ en ajoutant x nouvelles tâches à partir de l'instance initiale I. Nous ajoutons la contrainte de précédente suivante : chaque groupe de x (avec $x > \frac{(2c+6)c+(c+4)\rho n}{2c+6)-(2c+5)\rho}$) nouvelles tâches est un successeur des anciennes tâches (les anciennes tâches proviennent de la transformation polynomiale utilisée dans la preuve du Théorème 4.2.2). Nous obtenons ainsi un graphe complet entre les anciennes tâches et les nouvelles.

S'il existe un tel algorithme A, alors il pourrait être utilisé pour décider de l'existence d'un ordonnancement de longueur au plus $c + 4$.

Soit $A(I')$ (resp. $A^*(I')$) le résultat donné par A (resp. une solution optimale) sur l'instance I'.

1. Si $A(I') < (2c+5)\rho x + (c+4)\rho n$ alors $A^*(I') < (2c+5)\rho x + (c+4)\rho n$. Alors nous pouvons décider qu'il existe un ordonnancement sur l'instance I avec $C_{max} \leq c+4$. En effet, s'il n'existe pas un tel ordonnancement, alors une tâche de l'instance I est exécutée après l'instant $c+5$. Mais, peut-être c tâches sont exécutées après $2c+6$ et donc $A^*(I') > (2c+6)(x-c)$. Ceci est impossible, en effet si $(2c+6)(x-c) \geq (2c+5)\rho x + (c+4)\rho n$ alors $x \leq \frac{(2c+6)c+(4+c)\rho n}{2c+6)-(2c+5)\rho}$. Ceci est une contradiction avec les hypothèses. Une contradiction avec $x > \frac{(2c+6)c+(4+c)\rho n}{2c+6)-(2c+5)\rho}$. Ainsi, il existe un ordonnancement de longueur $c+4$ pour les anciennes tâches.
2. Nous supposons $A(I') \geq (2c+5)\rho x + (c+4)\rho n$. Alors, $A^*(I') \geq (2c+5)x + (c+4)n$ car l'algorithme A est une algorithme d'approximation avec performance garantie bornée par $\rho < \frac{2c+6}{2c+5}$. Il n'existe pas d'algorithme pour décider si les tâches à partir de l'instance I admette un ordonnancement de longueur au plus $c+4$.

   En effet, si il existe un tel algorithme, en exécutant les x tâches à l'instant $t = 4+2c$, nous obtenons un ordonnancement avec un temps de complétude strictement inférieur à $(5+2c)x + (4+c)n$ (il existe au moins une tâche qui est exécutée avant l'instant $t = c+4$). Ceci est une contradiction avec le fait que $A^*(I') \geq (2c+5)x + (c+4)n$.

   Ainsi, s'il existe un algorithme approché avec une performance garantie de $1 + \frac{1}{2c+5}$, alors cet algorithme peut être utilisé pour distinguer les instances positives (pour lesquelles la réponse est oui) des instances négatives (pour lesquelles la réponse est non) pour le problème d'ordonnancement $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max} = c+4$, produisant ainsi un algorithme polynomial pour un problème $\mathcal{NP}$-complet. Par conséquent, le problème $\bar{P}|prec; c_{.ij} = c \geq 2; p_i = 1|\sum C_j$ ne possède pas d'algorithme polynomial ρ-approché avec $1 + \frac{1}{2c+5}$.

□

4.2.4 Un algorithme polynomial pour $C_{max} = c+2$ avec $c \in \{2,3\}$

Théorème 4.2.4 *Le problème de décider si une instance de $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max}$ avec $c \in \{2,3\}$ admet un ordonnancement de longueur au plus $(c+2)$ peut-être résolu temps polynomial.*

Preuve

La preuve complète et détaillée et donnée dans [74].

□

La preuve proposée ne peut être étendue dans la cas général. En effet au dessus de la valeur $c = 4$ la combinatoire devient importante. Le challenge est de proposer un algorithme générique pour n'importe quelle valeur de $c = 2,\ 3, \ldots$

4.2.5 Approximation par dilatation

Après avoir établi une borne inférieure pour n'importe quel algorithme, le challenge est de proposer un algorithme approché avec garantie de performance non triviale. Dans cette partie, nous développons un algorithme approché pour le problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{\max}$.

4.2.5.1 Introduction, notation et description de l'algorithme

Notation : Nous notons par σ^{∞}, l'ordonnancement sur le modèle UET-UCT, et par σ_c^{∞} l'ordonnancement UET-LCT. De plus, nous notons par t_i (resp. t_i^c) le début d'exécution de la tâche i dans l'ordonnancement σ^{∞} (resp. dans l'ordonnancement σ_c^{∞}).
Principe : Nous gardons l'affectation des tâches sur les processeurs donnés par un « bon » ordonnancement sur un nombre illimité de processeurs σ^{∞}. Nous procédons à une dilatation de la longueur de l'ordonnancement pour préserver les délais de communication ($t_j^c \geq t_i^c + 1 + c$) pour deux tâches, i et j avec $(i, j) \in E$, exécutées sur deux processeurs différents.

Considérons un graphe de précédence $G = (V, E)$, nous déterminons un ordonnancement réalisable σ^{∞}, pour le modèle UET-UCT, en utilisant l'algorithme (de ratio 4/3) proposé par Munier et König [115]. Cet algorithme donne un couple $\forall i \in V,\ (t_i, \pi)$ pour ordonnancement σ^{∞} correspondant à : t_i est le début d'exécution de la tâche i pour l'ordonnancement σ^{∞} et π_i le processeur sur lequel la tâche i est ordonnancée à t_i.

Maintenant nous déterminons un couple $\forall i \in V,\ (t_i^c, \pi')$ pour l'ordonnancement σ_c^{∞} de la manière suivante : le début d'exécution $t_i^c = d \times t_i = \frac{(c+1)}{2} t_i$ et, $\pi = \pi'$. La justification du coefficient de dilatation est donnée ci-après. Une illustration de la dilatation est proposée à la figure 4.5.

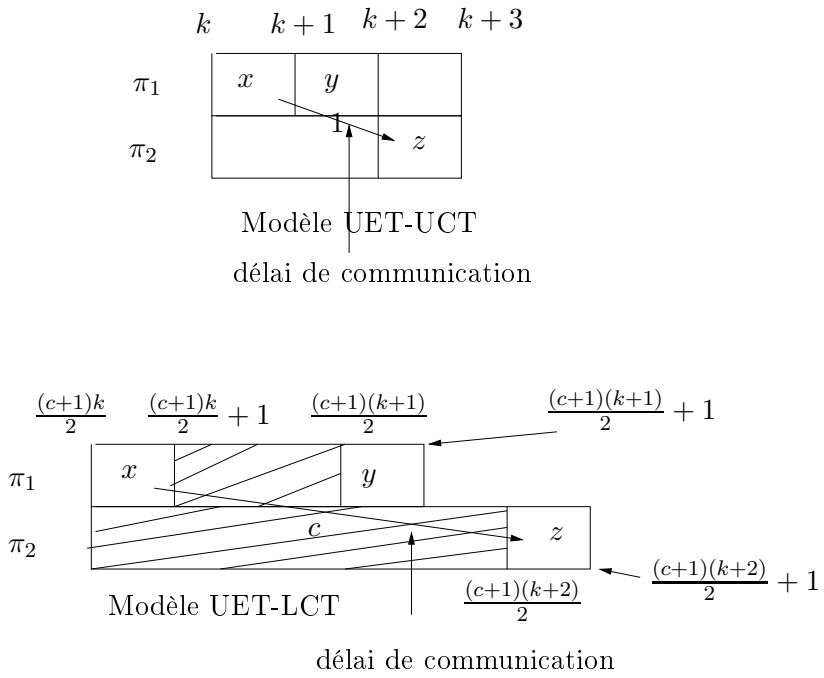


FIG. 4.5 – *Illustration de la notion de dilatation*

4.2.5.2 Analyse du ratio de performance

Lemme 4.2.2 *Le coefficient d'expansion est* $d = \frac{(c+1)}{2}$.

Preuve

Considérons deux tâches i et j telles que $(i,j) \in E$, qui sont exécutées sur deux processeurs différents dans un ordonnancement réalisable σ^{∞}. Nous cherchons un coefficient d tel que $t_i^c = d \times t_i$ et $t_j^c = d \times t_j$. Après l'expansion, dans le but de respecter les délais de communication et les contraintes de précédence nous obtenons $t_j^c \geq t_i^c + 1 + c$, et donc $d \times t_i - d \times t_j \geq c+1,\ d \geq \frac{c+1}{t_i - t_j},\ d \geq \frac{c+1}{2}$. Il suffit de choisir $d = \frac{(c+1)}{2}$.

□

Lemme 4.2.3 *L' algorithme d'expansion donne un ordonnancement réalisable pour le problème noté par* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{\max}$.

Preuve

Il suffit de vérifier que la solution donnée par l'algorithme d'expansion produit un ordonnancement réalisable pour le modèle UET-LCT. Considérons deux tâches i et j telles que $(i,j) \in E$. Nous notons par π_i (resp. π_j) le processeur sur lequel la tâche i (resp. la tâche j) est exécutée dans l' ordonnancement σ^{∞}. De plus, nous notons par π'_i (resp. π'_j) le processeur sur lequel la tâche i (resp. la tâche j) est exécutée dans l' ordonnancement σ_c^{∞}. Ainsi,

- Si $\pi_i = \pi_j$ alors $\pi'_i = \pi'_j$. Sachant que la solution proposée Munier et König [115] donne un ordonnancement réalisable sur le modèle UET-UCT, alors nous avons

$$t_i + 1 \leq t_j,\ \frac{2}{c+1}t_i^c + 1 \leq \frac{2}{c+1}t_j^c; t_i^c + 1 \leq t_i^c + \frac{c+1}{2} \leq t_j^c$$

.

- Si $\pi_i \neq \pi_j$ alors $\pi'_i \neq \pi'_j$. Nous avons

$$t_i + 1 + 1 \leq t_j,\ \frac{2}{c+1}t_i^c + 2 \leq \frac{2}{c+1}t_j^c; t_i^c + (c+1) \leq t_j^c$$

.

□

Théorème 4.2.5 *L'algorithme d'expansion donne un* $\frac{2(c+1)}{3}$ *algorithme approché pour le problème* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{\max}$.

Preuve

Nous notons par C_{max}^h (resp. C_{max}^{opt}) la longueur de l'ordonnancement déterminé par l'algorithme de Munier et König (resp. une solution donnée par un ordonnancement optimal σ^{∞}). De la même manière, nous notons par $C_{max}^{h^*}$ (resp. $C_{max}^{opt,c}$) la longueur de l'ordonnancement calculé par l'algorithme (resp. la valeur optimale de l'ordonnancement σ_c^{∞}).

Nous savons que $C_{max}^h \leq \frac{4}{3}C_{max}^{opt}$.

Ainsi, nous obtenons

$$\frac{C_{max}^{h^*}}{C_{max}^{opt,c}} = \frac{\frac{(c+1)}{2}C_{max}^h}{C_{max}^{opt,c}} \leq \frac{\frac{(c+1)}{2}C_{max}^h}{C_{max}^{opt}} \leq \frac{\frac{(c+1)}{2}\frac{4}{3}C_{max}^{opt}}{C_{max}^{opt}} \leq \frac{2(c+1)}{3}$$

. □

Machines	c_{ij}	C_{max}	Complexité	Seuil	Références
$\bar{P}$	$c = 1$	5	Polynomial		[172]
$\bar{P}$	$c = 1$	6	$\mathcal{NP}$-complet	$7/6 \leq \rho$	[172]
P	$c = 1$	3	Polynomial		[131]
P	$c = 1$	4	$\mathcal{NP}$-complet	$5/4 \leq \rho$	[172]
$\bar{P}$	$c \in \{2, 3\}$	$c + 2$	Polynomial		Théorème 4.2.4
$\bar{P}$	$c \geq 2$	$c + 4$	$\mathcal{NP}$-complet	$1 + 1/(c + 4) \leq \rho$	Théorème 4.2.2
P	$c \geq 2$	$c + 1$	Polynomial		[24]
P	$c \geq 2$	$c + 3$	$\mathcal{NP}$-complet	$1 + 1/(c + 3) \leq \rho$	[24]

TAB. 4.3 – Tableau 4.1 complété

4.3 Analyse des résultats

Nous complétons le tableau 4.1 par le tableau 4.3.

Maintenant, nous pouvons répondre par la négative concernant la question posée au début de la section bien que pour le moment nous ne connaissions pas encore la complexité pour $C_{max} = c + 3$ et pour $C_{max} = c + 2$ avec $c \geq 4$. Nous pouvons conclure que l'écart est différent. Le problème d'ordonnancement sur un nombre illimité de processeurs est aussi difficile en présence des grands délais de communication que sur un nombre limité. La difficulté provient des grands délais de communication et non pas du nombre limité ou non de processeurs.

Nous pouvons remarquer que pour le cas $c = 2$, nous avons prouvé que le problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{\max}$ est $\mathcal{NP}$-complet. Le résultat est identique à celui du problème classique $UET - UCT$. Rappelons que pour ce problème il existe un algorithme $\frac{4}{3}-$approché (basé sur une relaxation des contraintes d'intégrité d'un programme linéaire en nombres entiers avec une phase d'arrondis, voir [115]). En utilisant l'algorithme de l'expansion avec $c = 2$, nous obtenons une 2-approximation. Une question intéressante : *Est-ce que le programme linéaire en nombres entiers donné pour le problème UET-UCT en présence d'un nombre illimité de processeurs, peut il être étendu pour le problème $c = 2$?*

De plus, dans [88] les auteurs proposent une preuve de non-approximabilité, basée sur la technique du saut (voir [128]), pour le problème de la minimisation de la somme des temps de complétude.

Machines	c_{ij}	Approximation	Références
$\bar{P}$	$c = 1$	$\rho \leq 4/3$	[112]
P	$c = 1$	$\rho \leq 7/3$	[111]
$\bar{P}$	$c \geq 2$	2 pour les arbres	[110]
$\bar{P}$	$c \geq 1$	$\frac{2(c+1)}{3}$	Théorème 4.2.5
P	$c \geq 2$	$O(c)$	[140]

TAB. 4.4 – Nouveaux résultats d'approximation

4.4 Conclusion

Dans ce chapitre, nous avons étudié (du point de vue de la complexité et de l'approximation) deux problèmes d'ordonnancement dans le modèle homogène. La figure 4.6 complète les résultats de complexité.

A la vue des résultats d'approximation (voir la figure 4.7), il est légitime de se poser la question : « Est-ce qu'il existe un algorithme approché avec une garantie de performance de ρ avec $\rho \in \mathbb{N}$ pour les problèmes avec les grands délais de communication en présence d'un nombre fini ou infini de processeurs ? » Nous voulons aussi, démontrer l'appartenance de ces problèmes à la classe $\mathcal{APX}$ comme les problèmes pour lesquels la duplication est autorisé. Notons que si nous autorisosn la duplication il existe un algorithme 2-approché (voir [126]).

Nous avons également redémontrer le résultat classique de Hoogeveen *et al.* [172] en utilisant la résolution d'un système d'inéquations. De plus, dans [172], les auteurs ont montré la $\mathcal{NP}$-complétude pour le problème $UET - UCT$ dans le cas où le graphe de précédence admet pour structure une anti-arborescence. Il serait intéressant de proposer un seuil d'approximation pour le comparer à celui donné pour un graphe de précédence quelconque.

La borne de 4/3 obtenu par Munier et König [115] reste la meilleure borne connue depuis une quinzaine d'années. Une idée possible consiste à « découper » le graphe de précédence en des graphes G_i de profondeur au plus au plus cinq, sachant que décider si nous pouvons ordonnancer en temps cinq un G_i est polynomial. Cependant, la difficulté consiste à d'une part à générer une solution réalisable à partir de G_i et évidemment d'évaluer la performance relative.

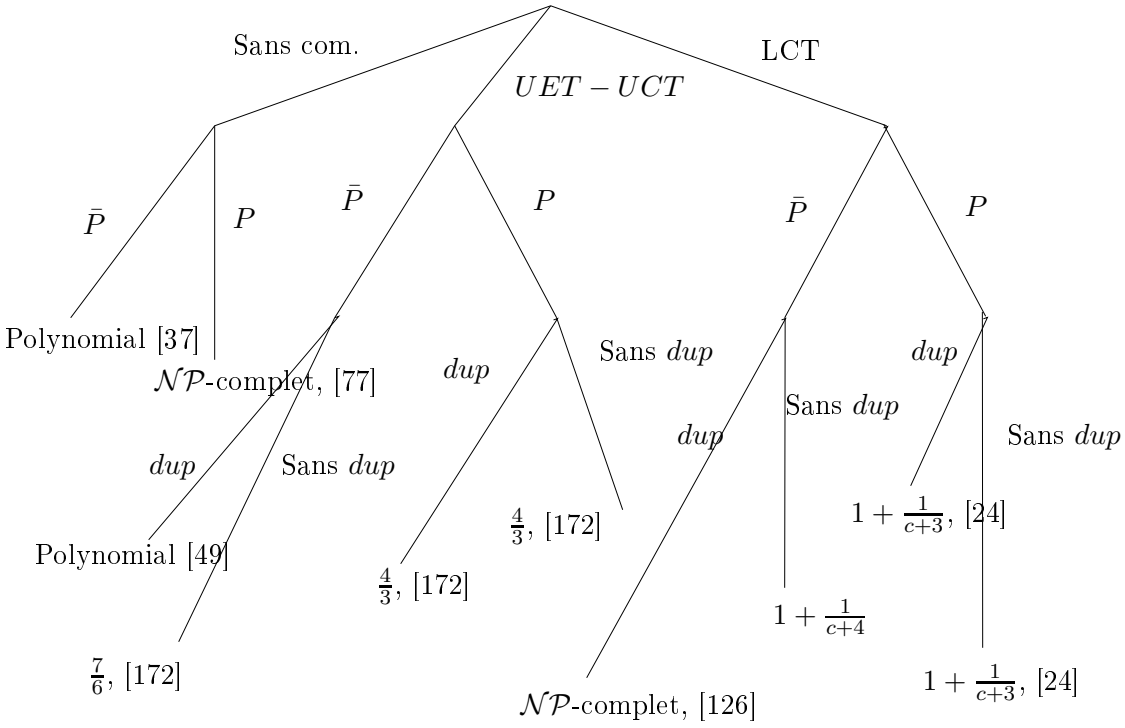


FIG. 4.6 – *Principaux résultats de complexité pour le modèle homogène pour la minimisation de la longueur de l'ordonnancement*

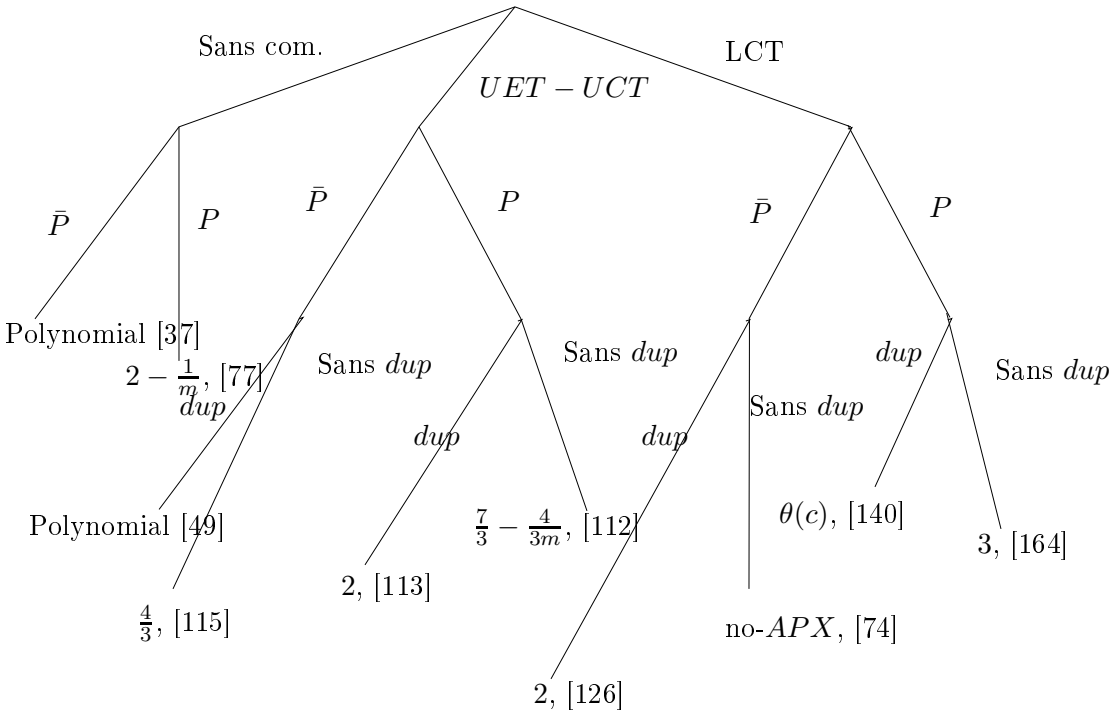


Fig. 4.7 – *Principaux résultats d'approximation pour le modèle homogène pour la minimisation de la longueur de l'ordonnancement*

CHAPITRE

5 Communications hiérarchiques

5.1 Présentation et motivations

5.1.1 Introduction

Dans ce chapitre, nous adoptons *le modèle à communications hiérarchiques* [27] dans lequel nous supposons que les délais de communications ne sont pas homogènes ; les processeurs sont regroupés sous forme de *modules* et les communications à l'intérieur d'un même module seront considérées comme plus rapides que celles entre deux processeurs appartenant à des modules différents. Ce modèle prend en compte la nature hiérarchique des communications dans les ordinateurs parallèles, composés de plusieurs réseaux de PC ou des stations de travail (NOW, Network of workstations) [129]. L'utilisation des réseaux de stations de travail comme machines parallèles [9, 129] a permis un regain d'intérêt pour le parallélisme mais a également engendré de nouveaux problèmes concernant l'exploitation potentielle de la puissance offerte par de tels systèmes.

La plupart des tentatives pour modéliser ces systèmes se sont focalisées sur la programmation de ces systèmes plutôt que le développement de modèles abstraits [142, 143, 40, 38]. Quelques tentatives concernant cette approche ont vu le jour dans la littérature [27, 46]. Le modèle que nous adoptons ici est le *modèle à communications hiérarchiques* qui est dédié à l'un des problèmes majeurs apparus du fait de la nécessité d'une utilisation efficace de telles architectures *le problème d'ordonnancement de tâches*. Le modèle proposé prend en compte les architectures des NOW : l'hypothèse des communications hiérarchiques *i.e.* une hiérarchie des délais de communications avec des latences de plus en plus grandes. Plus formellement, soit un ensemble de modules constitués de processeurs identiques, et un graphe de précédence, nous considérons que si deux tâches communicantes sont exécutées sur le même processeur (resp. sur des processeurs différents d'un même

module) alors le délai de communication correspondant est négligeable (resp. est égal ce que nous appellerons *le délai de communication inter-processeur*). A l'inverse, si ces tâches sont exécutées sur des modules différents, alors le délai de communication est plus important et celui-ci sera appelé *le délai communication inter-module*.

Nous disposons de m machines multiprocesseurs (ou modules) qui sont utilisées pour exécuter n tâches soumises à des contraintes de précédence. Chaque machine (module Π) comprend plusieurs processeurs parallèles identiques π. Un couple de délais de communications (c_{ij}, ϵ_{ij}) est associé à chaque arc (i, j) entre deux tâches dans le graphe de précédence. Par la suite, c_{ij} (resp. ϵ_{ij}) représentera les communications inter-module (resp. inter-processeur) et nous ferons l'hypothèse naturelle que $c_{ij} \geq \epsilon_{ij}$. Si les tâches i et j sont exécutées sur des machines différentes ($\Pi' \neq \Pi$), alors j devra être exécutée au moins c_{ij} unités de temps après la fin d'exécution de i. De manière similaire, si i et j sont exécutées sur la même machine ($\Pi = \Pi'$) mais sur des processeurs différents ($\pi \neq \pi'$) alors le début de l'exécution de la tâche j peut se faire ϵ_{ij} unités de temps après la fin d'exécution de i. De plus, si i et j sont exécutées sur le même processeur ($\pi = \pi'$) alors j peut avoir un début d'exécution immédiatement après la fin de la tâche i.

Formellement, dans le `modèle d'ordonnancement à communications hiérarchiques` nous associons un couple de valeur (c_{ij}, ϵ_{ij}) avec $\epsilon_{ij} \leq c_{ij}$ $\forall (i, j) \in E$ tel que :

- si $\pi = \pi'$ alors $t_i + p_i \leq t_j$
- sinon si $\pi \neq \pi'$ et si $\Pi = \Pi'$ alors $t_i + p_i + \epsilon_{ij} \leq t_j$
- sinon $\Pi \neq \Pi'$ et $t_i + p_i + c_{ij} \leq t_j$

où p_i (resp. t_i) désigne la durée d'exécution (resp. le début d'exécution) de la tâche i.

Clairement, le modèle à communications hiérarchiques est une généralisation du modèle à *communications homogènes* ([48], [50, 131]) puisque si $\forall (i, j) \in E,\ \epsilon_{ij} = c_{ij}$ alors, le modèle hiérarchique est identique au modèle classique avec *communications homogènes*.

En utilisant les notations $\alpha|\beta|\gamma$ (ces trois paramètres permettent de régler le niveau d'abstraction de la machine virtuelle et les hypothèses sur le type de graphes considérés), introduites par Graham *et al.* [77] et étendues dans [173], notre problème est représenté par $\bar{P}(P2)|prec; (c_{ij}, \epsilon_{ij}) = (2, 1); p_i = 1|C_{max}$ où le premier champ $\bar{P}(P2)$ signifie que nous disposons d'une infinité de machines constituées de deux processeurs identiques chacune et *prec* indique que le graphe orienté sans cycle admet une structure quelconque.

Le problème d'ordonnancement avec communications hiérarchiques a été

Modèle	Critère	Modèle homogène		Modèle hiérarchique	
		Type de machine	Borne(s)	Type de machine	Borne(s)
UET-UCT	C_{max}	$\bar{P}$	$\frac{7}{6} < \rho \leq \frac{4}{3}$, ([89],[115])	$\bar{P}(P2)$	$\frac{5}{4} \leq \rho \leq \frac{8}{5}$, ([30],[27])
SCT	C_{max}	$\bar{P}$	$\rho \leq \frac{2(1+\Phi)}{2\Phi+1}$, ([111])	$\bar{P}(P2)$	$\rho \leq \frac{12(1+\Phi)}{12\Phi+1}$, ([35])
UET-UCT	$\sum_j C_j$	$\bar{P}$	$\frac{9}{8} \leq \rho$, ([88])	$\bar{P}(P2)$	$\frac{7}{6} \leq \rho$, ([30])
UET-UCT,biparti	C_{max}	P	$\frac{5}{4} \leq \rho$, ([172])	$P2(P)$	$\frac{4}{3} \leq \rho$, ([11])
UET-UCT,biparti	$\sum_j C_j$	P		$P2(P)$	$\frac{17}{16} \leq \rho$, ([11])
UET-UCT	C_{max}	P	$\rho \leq \frac{7}{3} - \frac{4}{3m}$, ([112])	$P(Pm)$	$\rho \leq (3 - \frac{1}{Mm})$, ([11])
UET-UCT,dup	C_{max}	$\bar{P}$	$\rho = 1$, ([49])	$\bar{P}(P2)$	$\rho = 1$, ([25])

TAB. 5.1 – Tableau comparatif sur la complexité et sur la (non)-approximation des problèmes d'ordonnancement en présence de communications homogènes et hiérarchiques obtenus dans [68].

introduit dans [68]. Dans cette thèse, plusieurs résultats ont été obtenu, et sont résumés dans le tableau 5.1.

Cependant, plusieurs approches existent pour tenter de modéliser au mieux ces systèmes en prenant en compte les développements technologiques :

- une approche concerne la programmation système, nous pouvons citer les travaux référencés par [38, 40, 142, 143].
- dans une approche d'un modèle abstrait, nous pouvons citer les travaux de [39, 57, 59, 102, 107, 108, 168] sur les tâches malléables introduites dans [39, 57]. Une tâche malléable est une tâche qui peut s'exécuter sur plusieurs processeurs et sa durée d'exécution dépend du nombre de processeurs exécutant cette tâche.

En considérant le tableau 5.1, il est manifeste qu'un travail sur la complexité pour tout couple de valeurs de délais de communication est à effectuer. Ce travail a commencé à être fait dans [69], en considérant le couple de valeurs $(2, 1)$ et les principaux résultats de cet l'article sont donnés ci-après.

Théorème 5.1.1 *Le problème* $\bar{P}(P2)|prec;(c_{ij},\epsilon_{ij}) = (2,1);p_i = 1|C_{max}$ *ne possède pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à* $6/5$ *sous l'hypothèse* $\mathcal{P} \neq \mathcal{NP}$.

Théorème 5.1.2 *Le problème de décider si une instance de* $\bar{P}(P2)|prec;(c_{ij},\epsilon_{ij}) = (2,1);p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus trois est polynomial.*

Théorème 5.1.3 *Le problème* $\bar{P}(P2)|prec;(c_{ij},\epsilon_{ij}) = (2,1);p_i = 1|\sum_j C_j$ *ne possède pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à* $9/8$ *sous l'hypothèse* $\mathcal{P} \neq \mathcal{NP}$.

5.1.2 Motivations

Le challenge maintenant est double : « **Peut-on généraliser ces résultats pour tout couple de valeurs** (c, c') **avec pour fonction objective la minimisation de la longueur de l'ordonnancement ?** ». Dans le cas où la réponse est positive, « **Pour le problème** UET**, est-ce que l'écart (portant sur la valeur du** C_{max}**) entre l'existence d'une solution optimale (via un algorithme de complexité polynomiale) et la** $\mathcal{NP}$**-complétude, est identique entre le modèle à communications homogènes et celui à communications hiérarchiques en présence des grands délais de communication ?** » Dans ce chapitre, nous allons répondre à ces deux questions.

5.1.3 Présentation du chapitre

Le chapitre se décompose de la manière suivante :
- dans la section 5.2, nous établirons la $\mathcal{NP}$-complétude du problème pour la minimisation de la longueur de l'ordonnancement.
- dans la section 5.3, nous proposerons un algorithme polynomial pour $C_{max} = c + 1$.
- dans la section 5.4 nous effectuerons une analyse des résultats obtenus.
- dans la section 5.5 nous développerons un algorithme ρ-approché avec $\rho = \frac{2l(c+1)}{2l+1}$ (meilleure borne connue à notre connaissance).

Ces travaux ont été mené avec E. Angel, E. Bampis, J.C König, et A. Kononov (voir [11], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [69], [71], [72].

	Références	
(c_{ij}, ϵ_{ij})	C_{max}	$\sum_j C_j$
$(c \geq 3, 1)$	voir Théorème 5.2.1	voir Théorème 5.2.5
$(3, 2)$	voir Théorème 5.2.3	voir Théorème 5.2.5
$(c \geq 3, c')$	voir Théorème 5.2.2	voir Théorème 5.2.5
$(c, c-1)$	voir Théorème 5.2.4	voir Théorème 5.2.5

TAB. 5.2 – Résultats de complexité pour la section 5.2

5.2 Résultats sur la non-approximabilité pour tous couples de valeurs

Dans cette section, plusieurs preuves de $\mathcal{NP}-$complétude sont proposées. Nous montrons que le problème de décider si une instance de $\bar{P}(Pl \geq 4)|prec; (c_{ij},\ \epsilon_{ij}) = (c \geq 3, 1); p_i = 1|C_{max}$ possède un ordonnancement de longueur $(c+3)$ est $\mathcal{NP}$-complet (voir Théorème 5.2.1). Ce résultat sera généralisé dans le Théorème 5.2.2 pour le problème $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c, c'); p_i = 1|C_{max}$ avec $c > c' + 1 > 2$.

Nous montrons également la $\mathcal{NP}-$complétude pour le problème $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (3, 2); p_i = 1|C_{max}$ (voir Théorème 5.2.3). La transformation polynomiale est basée sur la $\mathcal{NP}-$complétude du problème $\mathcal{P}_1$, mais la construction de l'instance est différente. Nous tirons avantage des différentes valeurs des couples de communications. Ce résultat est également généralisé par le Théorème 5.2.4 les couples de valeurs : $(c_{ij}, \epsilon_{ij}) = (c, c-1)$ avec $c \geq 3$.

Dans un second temps, nous étudierons le problème avec pour fonction objectif en considérant la somme des temps de complétude $\sum_j C_j$, au lieu de la longueur de l'ordonnancement C_{max}.

Tous les résultats de $\mathcal{NP}-$complétude sont donnés par le tableau 5.2. Dans toutes les preuves de $\mathcal{NP}-$complétude la valeur des variables l et c sont constantes.

5.2.1 Minimisation de la longueur de l'ordonnancement

Théorème 5.2.1 *Le problème de décider si une instance de* $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c \geq 3, 1); p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus* $(c+3)$ *est* $\mathcal{NP}$*-complet.*

Preuve

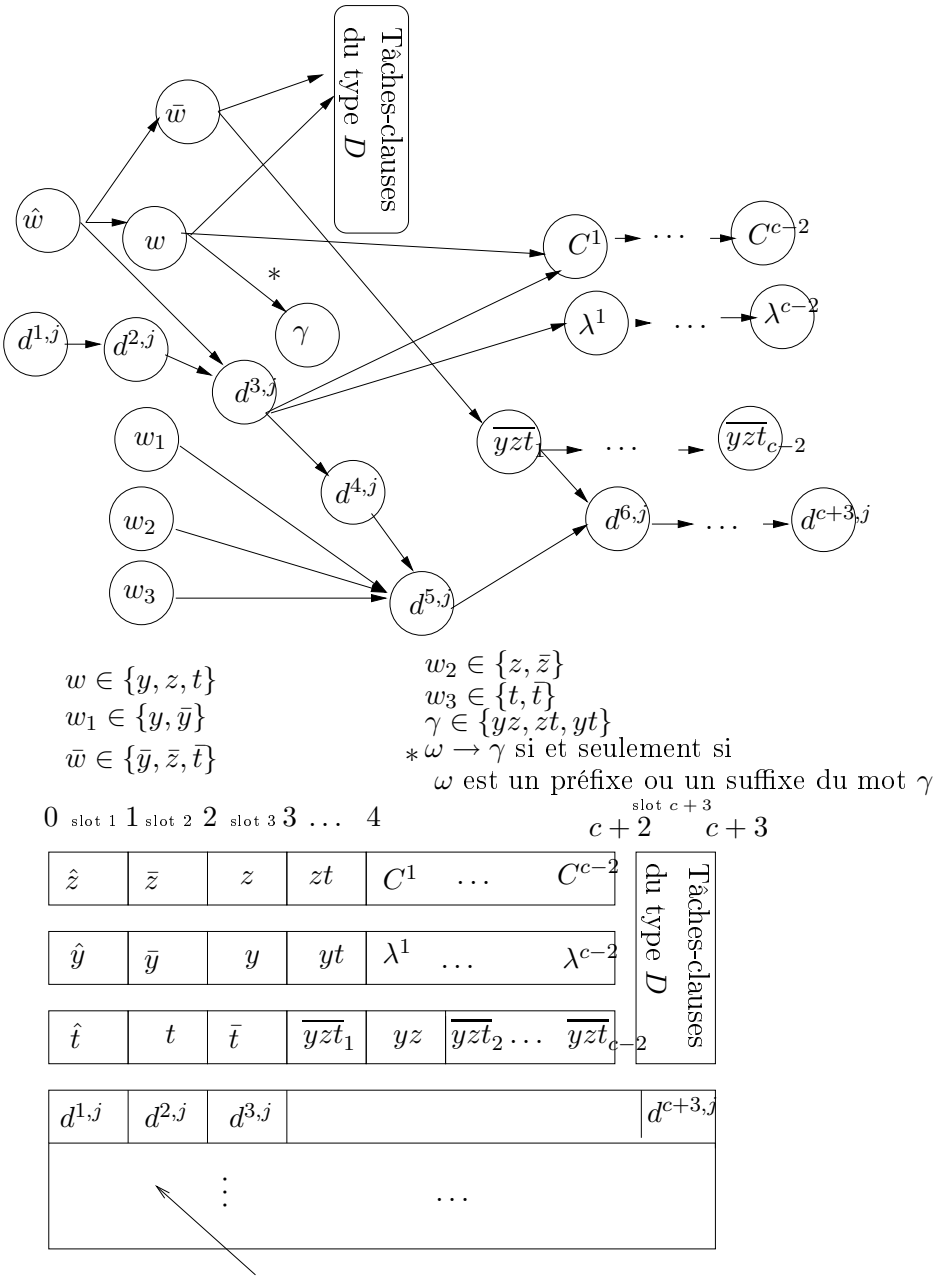


FIG. 5.1 – *Ordonnancement partiel sur un cluster K_i, et sous-graphe partiel du graphe de précédence pour une clause $C = (y \vee z \vee t)$ pour le problème $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c \geq 3, 1)|C_{max}$*

Il est facile de voir que $\bar{P}(Pl \geq 4)|prec;(c_{ij},\epsilon_{ij}) = (c \geq 3,1);p_i = 1|C_{max} = c+3 \in \mathcal{NP}$.

Montrons la $\mathcal{NP}-$complétude.

Notre preuve est basée sur une réduction à partir de $\mathcal{P}_1$

Considérons une instance π^* of $\mathcal{P}_1$, nous construisons une instance π du problème $\bar{P}(Pl \geq 4)|prec;(c_{ij};\epsilon_{ij}) = (c \geq 3,1);p_i = 1|C_{max} = c+3$ de la manière suivante : (voir la figure 5.1) :

Remarque

Dans le but de ne pas surcharger la figure 5.1, nous présentons de manière synthétique le sous-graphe partiel du graphe de précédence qui est construit à partir de la clause $C = (y \vee z \vee t)$. Quand nous écrivons $\hat{w} \rightarrow \bar{w}$ avec $w \in \{y,z,t\}$, ceci signifie que nous avons trois chaînes de longueur deux (une pour chaque variable booléenne). De manière similaire, quand nous écrivons $\omega \xrightarrow{*} \gamma$ avec $\gamma \in \{yz,zt,yt\}$ et ω est un préfixe ou un suffixe du mot γ, nous avons par exemple, si $\omega = y$, $y \rightarrow yz$ et $y \rightarrow yt$.

1. Pour chaque variable $\omega \in \mathcal{V}$ (où $\mathcal{V}$ représente l'ensemble des variables dans la formule logique), nous introduisons trois tâches ω, $\bar{\omega}$ et $\hat{\omega}$, appelé tâches-variables, et deux contraintes de précédence : $\hat{\omega} \rightarrow \omega$ et $\hat{\omega} \rightarrow \bar{\omega}$.
2. Pour chaque clause $C = (x \vee \bar{y})$ de longueur deux, nous introduisons une tâche D (**ces tâches dans la suite seront appelés des tâches-clauses de type D**). Nous ajoutons les contraintes de précédence suivantes : $x \rightarrow D$ et $\bar{y} \rightarrow D$.
3. Pour chaque clause $C = (y \vee z \vee t)$ de longueur trois,
   (a) nous introduisons $(2 \times (c-2)+3)$ tâches-clauses, $\gamma \in \{yz,yt,zt\}, C^k$ et $\overline{yzt}_k$ avec $k \in \{1,\ldots,c-2\}$. Pour chaque variable booléenne ω apparaissant dans C, nous ajoutons les contraintes suivantes :
   $\omega \rightarrow C^1,\ \omega \rightarrow \gamma$ si $\omega \in \gamma$ où $\gamma \in \{yz,yt,zt\}$ et

$$C^k \rightarrow C^{k+1}, \text{ et } \overline{yzt}_k \rightarrow \overline{yzt}_{k+1},\ k \in \{1,\ldots,c-3\}$$

   (b) Nous ajoutons $((c+3) \times (l-3)+(c-2))$ tâches « dummy » [1] notées $d^{k,j},\ \forall j \in \{1,\ldots,l-3\},\ k \in \{1,\ldots,c+3\}$ et λ^p avec $p \in \{1,\ldots,c-2\}$ dont les contraintes de précédence sont :

[1] Dans le reste de ce chapitre les tâches « dummy » $d^{k,j},\ \forall j \in \{1,\ldots,l-3\},\ \forall k \in \{1,\ldots,c+3\}$ sont associées à la clause C de taille trois. Donc, si le nombre de clauses de taille trois est q, dans l'instance construite nous avons $q \times (l-3) \times (c+3)$ tâches « dummy ».

- $d^{k,j} \rightarrow d^{k+1,j},\ \forall j \in \{1,\ldots,l-3\},\ k \in \{1,\ldots,c+2\}$ et $\lambda^p \rightarrow \lambda^{p+1},\ p \in \{1,\ldots,c-3\}$.
- Nous ajoutons également, $d^{3,j} \rightarrow C^1,\ \omega \rightarrow d^{5,j},\ \bar{\omega} \rightarrow d^{5,j},\ d^{3,j} \rightarrow \lambda^1,\ \hat{\omega} \rightarrow d^{3,j},\ \forall j \in \{1,\ldots,l-3\}$ où ω désigne une variable booléenne apparaissant dans la clause C.
- En dernier lieu, pour chaque variable booléenne ω apparaissant dans la clause C, nous ajoutons les contraintes de précédentes suivantes : $\bar{\omega} \rightarrow \overline{yzt}_1 \rightarrow d^{6,j},\ \forall j \in \{1,\ldots,l-3\}$.

La construction présentée ci-dessus est illustrée par la figure 5.1. La transformation est clairement calculable en temps polynomial.

- Supposons maintenant qu'il existe un ordonnancement de longueur au plus $(c+3)$. Nous allons montrer qu'il existe une affectation $I : \mathcal{V} \rightarrow \{0,1\}$ des valeurs de vérité aux variables satisfaisant l'instance π^* du problème $\mathcal{P}_1$.
  Le ième cluster sera noté par K_i.
  Dans ce cas, nous pouvons remarquer que :
  Soit $C = (y \vee z \vee t)$ une clause.

  1. Toutes les tâches d'un chemin de longueur quatre (la longueur d'un chemin est le nombre de tâches constituant ce chemin) sont exécutées sur le même cluster. Donc, les tâches C^k, $d^{k,j}$, $\overline{yzt}_k$, ω, $\bar{\omega}$, $\hat{\omega}$ avec $\omega \in \{y,z,t\}$ sont ordonnancées sur un cluster K_i.
  2. Les tâches « dummy » $d^{k,j}$ utilisent $(l-3)$ processeurs de K_i sans tant d'inactivité (une communication n'est pas autorisée sur un chemin de longueur $(c+3)$).
  3. Les contraintes de précédence entre les prédécesseurs de tâche $d^{5,j}$ (*i.e.* ω et $\bar{\omega}$ avec $\omega \in \{y,z,t\}$) impliquent que ces prédécesseurs sont exécutés aux slots 1, 2 et 3 sur trois autres processeurs.
  4. Les contraintes de précédence entre les tâches $\overline{yzt}_1$ et $d^{6,j}$ impliquent que la tâche $\overline{yzt}_1$ est affectée au slot 4 et donc deux tâches parmi $\bar{y}$, $\bar{z}$, $\bar{t}$ sont exécutées au slot 2.
  5. Au slot 2, une tâche parmi $y,\ z,\ t$ doit être exécutée. Autrement, les tâches C^k, λ^k, γ, avec $\gamma \in \{yt, yz, zt\}$, $\overline{yzt}_j$, $j > 1$ et les trois tâches-clauses de type D admettant les tâches y,z,t comme prédécesseurs doivent exécutées durant les slots 5 à $(c+3)$ sur trois processeurs libres de K_i. Sachant qu'au plus $(3c-3)$ tâches peuvent être affectées sur trois processeurs durant $(c-1)$ slots, ceci est impossible.

Supposons que les tâches $\bar{y}$, $\bar{z}$ et t sont affectées au slot 2. Les tâches-clauses de type D, admettant les tâches y, z et $\bar{t}$ comme prédécesseurs sont exécutées au slot $(c+3)$ sur le même cluster car elles ne peuvent être ordonnancées sur un autre cluster (sachant les communications inter-cluster sont de valeurs c) et les tâches correspondant aux autres littéraux sont exécutées au slot 2 sur un autre cluster.
Maintenant, nous affectons la **valeur vraie** au littéral si la tâche associée est exécutée **au slot 2** et **faux sinon**.
Ceci donne une solution à note problème.

- Supposons qu'il existe une affectation $I : \mathcal{V} \rightarrow \{0,1\}$ des valeurs de vérité aux variables satisfaisant l'ensemble des clauses. Supposons que le littéral dans la clause $C = (y \vee z \vee t)$ est t. Donc, l'ordonnancement proposé à la figure 5.1 est réalisable sur trois processeurs libres.

Ceci conclut la preuve du Théorème 5.2.1.

□

Le résultat précédent peut-être généralisé par les Théorèmes suivants 5.2.2. Dans les trois prochains Théorèmes, nous donnerons seulement la transformation polynomiale. Les démonstrations sont similaires à celle proposée dans la preuve du Théorème 5.2.1.

Théorème 5.2.2 *Le problème de décider si une instance de* $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c, c'); p_i = 1|C_{max}$, *avec* $c > c' + 1 > 2$ *possède un ordonnancement de longueur au plus* $(c+3)$ *est* $\mathcal{NP}$*-complet.*

Théorème 5.2.3 *Le problème de décider si une instance de* $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (3, 2); p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus six est* $\mathcal{NP}$*-complet.*

Théorème 5.2.4 *Le problème de décider si une instance du problème* $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c, c-1); p_i = 1|C_{max}$ *avec* $c \geq 3$, *possède un ordonnancement de longueur au plus* $(c+3)$ *est* $\mathcal{NP}$*-complet.*

Corollaire 5.2.1 *Il n'existe pas d'algorithme approché pour le* $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c, c'); p_i = 1|C_{max}$ *avec* $c > c'$ *avec pour performance* $1 + \frac{1}{c+3}$.

Preuve

Le Corollaire 5.2.1 est une conséquence immédiate du Théorème de l'impossibilité, (voir [50], [67]) et Théorèmes 5.2.1, 5.2.3, 5.2.2, 5.2.4.

Ceci conclut la preuve du corollaire 5.2.1.

□

5.2.2 La minimisation de la somme des temps de complétude

Dans cette section, nous allons montrer qu'il n'existe pas d'algorithme d'approximation pour le problème $\bar{P}(Pl \geq 4)|prec;(c_{ij},\epsilon_{ij}) = (c \geq 3,c');p_i = 1|\sum_j C_j$, avec $c > c'$, avec une performance relative inférieure à $1+\frac{1}{2c+4}$ sous $\mathcal{P} \neq \mathcal{NP}$.

Théorème 5.2.5 *Il n'existe pas d'algorithme d'approximation pour le problème* $\bar{P}(Pl \geq 4)|prec;(c_{ij},\epsilon_{ij}) = (c \geq 3,c');p_i = 1|\sum_j C_j$ *avec* $c > c'$ *avec un ratio inférieure à* $1+\frac{1}{2c+4}$.

Preuve

La preuve du Théorème 5.2.5 est similaire à celle donné pour le Théorème 4.2.3 avec pour valeur de $x > \frac{(2c+5)lc+(c+3)\rho n}{(2c+5)-(2c+4)\rho}$.

Nous appliquons la même méthode que celle proposée pour la démonstration du Théorème 4.2.3. □

5.3 Un algorithme polynomial pour $C_{max} = c+1$

Théorème 5.3.1 *Le problème de décider si une instance de* $\bar{P}(Pl)|prec;(c_{ij},\epsilon_{ij}) = (c > 0,c');p_i = 1|C_{max}$ *(resp.* $\bar{P}(Pl \geq 3)|prec;(c_{ij},\epsilon_{ij}) = (c \geq 3,1);p_i = 1|C_{max}$*) possède un ordonnancement de longueur au plus* $(c+1)$ *est polynomial sachant* l *et* c *sont constants.*

Preuve

En effet, dans le où $C_{max} = c+1$ les communications inter-clusters sont interdites. Donc, chaque composante connexe du graphe de précédence doit être constitué d'au plus $l \times (c+1)$ tâches. Le problème de déterminer si un graphe de taille au plus $l \times (c+1)$ peut être ordonnancé en $c+1$ unités de temps est clairement polynomial quand l et c sont des constantes.

□

5.4 Discussion

Avec les résultats précédent, la réponse à la question émise au début de ce chapitre est donnée par la figure 5.2. Sachant que le modèle hiérarchique

que nous considérons ici et une généralisation du modèle classique d'ordonnancement, dans les deux cas (UET-UCT et UET-LCT) le problème devient plus difficile.

De plus, pour $c \geq 4$ le problème a été montré plus facile pour le modèle hiérarchique. En effet, dans [74], nous avons montré qu'il n'existe de ρ-approximation avec $\rho < 1 + 1/(c+4)$ pour le problème $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ et pour $C_{max} = c+1$ le problème est polynomial, et pour $C_{max} = c+2$ le problème est partiellement ouvert. Pour $C_{max} = c+3$ la complexité du problème est inconnue. Nous conjecturons à partir de la figure 5.2 que le problème de décider si une instance de $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ est ordonnançable pour $C_{max} = c+3$ est polynomial.

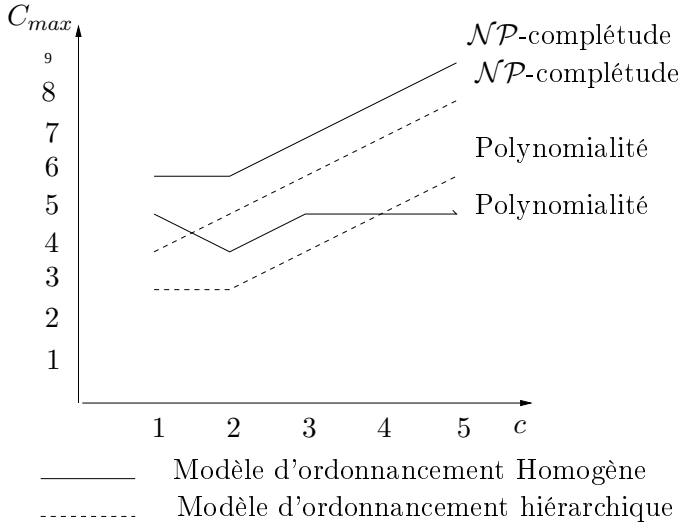


FIG. 5.2 – *Résultats de complexité pour le problème UET-UCT dans le cas du pire cas*

5.5 Approximation

Théorème 5.5.1 *L'algorithme d'expansion proposé dans le chapitre 4 donne un algorithme approché de ratio* $\frac{2l(c+1)}{2l+1}$ *pour le problème* $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (c \geq 3, c'); p_i = 1|C_{max}$.

Preuve

En utilisant l'algorithme générique proposé dans le chapitre 4 et le ré-

sultat connu (voir [27] pour l'approximation de $\bar{P}(Pl \geq 4)|prec; (c_{ij}, \epsilon_{ij}) = (1,0); p_i = 1|C_{max}$) nous obtenons le résultat souhaité.

□

5.6 Conclusion

Nous avons réussi à proposer un résultat général sur le seuil d'approximation (resp. un algorithme approché) pour n'importe quel algorithme approché pour n'importe quel couple de valeurs de délais de communication inter/intra cluster (c, c'). Il est important de noter que les techniques et les méthodes utilisées pour la recherche de solutions approchées et la classification au sens de la complexité restent valables. Cependant, la gamme des problèmes pouvant être étudiés est plus restreinte car nous nous heurtons très rapidement à une combinatoire importante (par exemple il n'existe pas pour le moment de résultat pour les petits délais de communications sur une infinité de clusters en présence de la duplication).

Nous avons proposé un modèle qui capture la notion de communications hiérarchiques et qui est une généralisation du modèle homogène. Ce modèle est un modèle plus complexe que le modèle homogène, mais la complexité reste raisonnable car nous avons pu effectuer une analyse analytique ($\mathcal{NP}$-complétude, algorithmes polynomiaux, algorithmes d'approximation), et ne se limite pas à une analyse stochastique ou statistique.

CHAPITRE

6 Communications locales

6.1 Présentation et motivations

6.1.1 Introduction

Dans un soucis de proposer des modèles d'ordonnancement « le plus proche de la réalité », nous présentons un modèle, que nous appellerons par la suite `modèle d'ordonnancement homogène avec des délais de communications locaux`. Ce modèle est une autre généralisation du `modèle à délais de communication homogènes`, comme le modèle hiérarchique présenté au chapitre 5. Dans ce modèle (à délais de communications locaux) l'allocation des tâches sur les processeurs n'a aucune influence sur la longueur de l'ordonnancement. Lorsqu'il existe une communication entre deux tâches i et j, alors i et j peuvent être alloué sur n'importe quel processeur. La valeur du délai de communication est fixe et est donné par le graphe de précédence. En effet, le graphe de processeurs (noté par la suite $G^* = (V^*, E^*)$ où $V^* = \{\pi_1, \ldots, \pi_m\}$ est un ensemble de m processeurs et E^* est l'ensemble des relations entre les processeurs, est totalement connecté, alors le début d'exécution de i dépend simplement du délai de communication potentiel entre i et ces prédécesseurs.

Dans le modèle présenté, nous relâchons la contrainte sur le caractère complet du graphe de processeurs. Alors, l'allocation des tâches sur les processeurs devient une caractéristique fondamentale[1] et essentielle pour ce modèle. Nous considérons un modèle pour lequel une fonction de distance est prise en compte entre deux processeurs π et π' dans le graphe de processeurs. Cette fonction influence le délai de communication entre deux tâches i et j (soumises à une contrainte de précédence), et donc sur la date de début d'exécution de la tâche j. Ainsi le délai de communication, utilisé pour le

[1] Cette caractéristique se retrouve dans le modèle des tâches malléables et modelables évoquée brièvement dans le chapitre 5.

calcul des dates de début d'exécution, entre deux tâches i et j soumises à des contraintes de précédence dépend maintenant de deux paramètres :

- le délai de communication c_{ij} issu du graphe de précédence G,
- la distance $d_{G^*}(\pi^l, \pi^h)$, entre les processeurs π^l et π^h exécutant les tâches i et j

Le temps de communication sera donné par la variable c_{j,π^l,k,π^h}. La valeur de la variable indique que le temps de la communication entre la tâche j, qui est exécutée sur le processeur π^l, et la tâche k, qui est ordonnancée sur le processeur π^h) sera défini par $c_{jk}d(\pi^l, \pi^h)$ où c_{jk} est le délai de communication issu du graphe de précédence et $d(\pi^l, \pi^h)$ est la fonction de distance.

La fonction de distance $d(\pi^l, \pi^h)$ mesure la distance entre les processeurs π^h et π^i dans le graphe de processeurs. Il est important de noter que la fonction de distance peut posséder des spécifications selon le problème d'ordonnancement étudié. La première fonction de distance a été proposé par Picouleau [130], et était définie par $d(\pi^l, \pi^h) = |l - h|$ (si sur la chaîne π^l (resp. π^k) est le $l$ème (resp. $k$ème) sommet). Cette fonction est appropriée pour les réseaux linéaires. Pour les autres réseaux structurés, la fonction de distance privilégiée est celle qui donne la longueur d'un plus court chemin pour chaque paire de processeurs.

Formellement, nous avons :

$$\forall (i,j) \in E, t_j \geq t_i + p_i + c_{ij} d_{G^*}(\pi^l, \pi^h)$$

où,

- π^l est le processeur où la tâche i est exécutée (la date de début d'exécution de la tâche i ne dépend pas du processeur sur lequel la tâche i est affectée),
- $d_{G^*}(\pi^l, \pi^h)$ est la longueur d'un plus court chemin dans G^* entre deux processeurs exécutant i et j (noté dans la suite $d(\pi^l, \pi^h)$),
- c_{ij} est le délai de communication donné par le graphe de précédence,
- p_i est la durée d'exécution de la tâche i,
- t_i est la date de début d'exécution de i.
- π^l (resp. π^h) est le processeur exécutant i (resp. j).

Le modèle exprime le fait que si deux tâches i et j, avec $(i,j) \in E$ alors la distance entre les processeurs π^l et π^h sur lesquels les tâches i et j sont executées doit être en compte dans le calcul de la date de début d'exécution de la tâche j. Alors, plus les processeurs π^l et π^h sont proches plus la date pourra être potentiellement avancée. Ainsi, le délai de communication qui devra être utilisé pour le calcul des dates de début d'exécution est propor-

tionnel aux délais de communication issu du graphe de précédence et de la fonction de distance.

Il est important de noter qu'avec une telle généralisation les liens qui relient les processeurs admettent une capacité infinie. Nous supposerons que nous n'avons pas de problème de congestion. Et donc, la transmission de plusieurs données sur même un lien ne peut conduire à un conflit.

Remarquons que `le modèle à délais de communications locaux` que nous considérons ici est une généralisation du modèle classique d'ordonnancement présenté dans le chapitre 4, [48], [50]. Considérons une instance telle que toutes les distances dans le graphe de processeurs sont unitaires, $d(\pi^l, \pi^h) = 1, \forall i, j$ (G^* est donc un graphe complet). Dans ce cas là, le modèle considéré est exactement le modèle à délai de communication homogène.

Dans le but d'illustrer le modèle, nous considérons l'exemple donné la figure 6.1. Nous prenons un graphe G constitué de cinq tâches, de durée d'exécution unitaires et une chaîne de processeurs de longueur quatre. Nous allons produire deux ordonnancements. Dans l'ordonnancement a), la tâche x (resp. d) est alloué sur le processeur π^1 (resp. π^4). Si la tâche e est allouée au processeur π^4 nous avons $\max\{d(\pi^4, \pi^h)\} = 3$ où π^h désigne le processeur sur lequel un prédécesseur de e est exécutée. Alors, $t_e \geq 4$, et nous obtenons $t_e = 4$. A contrario, si la tâche e est allouée au processeur π^3, nous avons $\max\{d(\pi^3, \pi^h)\} = 2$, alors $t_e \geq 3$. L'ordonnancement optimal est présenté en c).

Utilisant les trois champs Graham et al [77], si nous considérons le problème de la minimisation de la longueur de l'ordonnancement pour des tâches unitaires et des commnications unitaires (données par le graphe de précédence), et des communication unitaire (les délais de communication dépend d'un plus court chemin dans le graphe G^*) sur un ensemble de processeurs représenté par une chaîne, ce problème sera noté par $\{P, étoile\}|prec; c_{ij} = d(\pi^l, \pi^h); p_i = 1|C_{max}$.

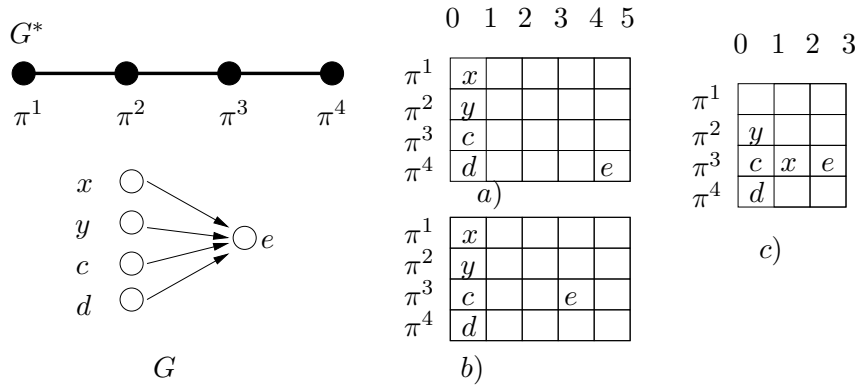


FIG. 6.1 – *Illustration de l'influence de l'allocation des prédécesseurs d'une tâche*

Dans la suite de ce chapitre, nous considérons le problème d'ordonnancement classique $UET - UCT$ (Unit Execution Time-Unit Communication Time, i.e. $\forall i \in V, p_i = 1$, and $\forall (i,j) \in E, c_{ij} = 1$) sur un nombre de processeurs limités. Les processeurs sont numérotés de π^0, $\pi^1, \ldots,$ π^{n-1} et le processeur π^k peut communiquer avec le processeur $\pi^{k'}$ avec un coût de communication de $d(\pi^k, \pi^{k'})$ où $d(\pi^k, \pi^{k'})$ représente un plus court chemin dans le graphe G^* entre ces deux processeurs. Ainsi la délai de communication est la fonction de distance décrite ci-avant.

6.1.2 Présentation du chapitre

Le chapitre se décompose de la manière suivante :

- dans la section 6.2, nous procéderons à un rappel des résultats de complexité et d'approximation obtenus sur ce modèle et nous rappellerons également un résultat classique qui permet de déterminer la performance relative d'un algorithme approché sur un nombre fini de processeurs à partir de celle obtenue sur un nombre illimité.
- dans la section 6.3, nous présentons nos premiers résultats en complexité et en approximation portant sur l'anneau de processeurs.
- dans la section 6.4, nous étudions la topologie en étoile ($\mathcal{NP}-$complétude section 6.4.2, algorithme polynomiale section 6.4.3 et un algorithme d'approximation avec garantie de performance non triviale section 6.4.5).
- dans la section 6.5, nous proposons une classe de graphes $\mathcal{G}$ pour laquelle le problème de décider si une instance de $\{P, p \in \mathcal{G}\}|prec, c_{ij} = d(\pi^l, \pi^h), p_i = 1|C_{max}$ possède un ordonnancement de longueur au plus

deux (resp. trois) est polynomial (resp. $\mathcal{NP}$-complet), même dans le cas où le graphe de précédence est un graphe biparti. Nous développons également un algorithme d'approximation générique pour un graphe de processeurs ayant un diamètre constant.

Ces travaux ont été mené avec V. Boudet,M. Bouznif, J. Cohen, J.C. König, B. Valéry, (voir [42], [44], [75]).

6.2 Résultats existants en complexité et approximation

Ce modèle a été introduit en premier par Picouleau dans [130]. Dans cet article l'auteur propose plusieurs résultats de $\mathcal{NP}$-complétude pour ce modèle :

- dans le cas où le graphe de précédence est une anti-arborescence (resp. arborescence) sur une topologie ayant pour structure une chaîne illimitée (resp. une chaîne illimitée et une étoile),

De plus, Lahlou dans [96], précise que le problème de décider sur un anneau ou une chaîne un graphe de précédence quelconque est $\mathcal{NP}-$complet pour $C_{max} = 3$. Il propose un algorithme de liste (proposé par Rayward-smith [141]) avec une performance relative ρ telle que $\lceil\sqrt{m}\rceil \leq \rho \leq 1 + \frac{3}{8}m - \frac{1}{2m}$. Dans [42] le problème d'ordonnancement dans lequel le graphe de processeurs est une chaîne ou un anneau a été étudié. Nous avons prouvé qu'il n'existe pas d'algorithme approché avec une performance relative de 4/3 pour la minimisation de la longueur de l'ordonnancement. Nous avons étendu ce résultat au cas où le graphe de précédence est un graphe biparti.

D'autres travaux (analyse d'algorithmes d'approximation basés sur la notion de listes dans [91], prise en compte de la contention dans [61] et [132], ou de la taille des données dans [60]) ont été menés sur ce type de modèle.

6.2.1 Motivations

Deux questions fondamentales émergent avec ce modèle d'ordonnancement : « **Est-ce que nous pouvons déterminer des seuils d'approximation pour les diverses topologies ?** » et « **Pour un problème classifié $\mathcal{NP}-$complet, pouvons-nous proposer une ρ-approximation avec $\rho \in \mathbb{N}$?** »

Le tableau 6.1 récapitule les résultats antérieurs sur ce modèle. Nous

G^*		G	Complexité	Référence
Topologie	$d(\pi^l, \pi^h)$			
Chaîne illimitée	1	$arborescence$	$\mathcal{NP}$-complet	[130]
	1	$anti-arbo$	$\mathcal{NP}$-complet	
Étoile	1	$arborescence$	$\mathcal{NP}$-complet	[130]
Cycle	1	$prec$	$\rho \geq 4/3$	[96]
Chaîne	1	$prec$	$\rho \geq 4/3$	[96]

TAB. 6.1 – Résumé des résultats de complexité antérieurs montrés sur ce modèle

pouvons noter qu'il n'existe pas de résultat de non-approximabilité. Nous proposerons dans la suite plusieurs résultats sur le seuil d'approximabilité pour plusieurs topologies de graphes de processeurs.

Dans toute la suite de ce chapitre sauf mention explicite, nous condidérons le problème central de l'ordonnancement c'est à dire $UET - UCT, \forall i, P_i = 1, \forall (i,j) \in E, c_{ij} = 1$. Le coût de communication dépend seulement du placement des tâches sur les processeurs.

6.2.2 Le principe du pliage en ordonnancement

Dans cette section, nous présentons un algorithme simple qui produit un ordonnancement σ^m sur m machines à partir d'un ordonnancement σ^∞ sur un nombre infini de processeurs pour $\bar{P}|prec, c_{ij} = 1, p_i = 1|C_{max}$. La validité de cet algorithme provient du fait qu'il existe au plus un couplage entre les tâches exécutées à t_i et celles exécutées à $t_i + 1$ dans l'ordonnancement σ^∞ (voir [165]). Nous utiliserons ce résultat dans la section 6.4.5.

Algorithme 6.1 Ordonnancement sur m processeurs à partir d'un ordonnancement σ^∞ sur un nombre infini

pour $i = 0$ à $C^\infty_{max} - 1$ **faire**
  Soit X_i l'ensemble des tâches ordonnancées à t_i dans σ^∞ en utilisant l'heuristique h^*.
  Les X_i tâches sont exécutées en $\lceil \frac{|X_i|}{m} \rceil$ unités de temps.
fin pour

Théorème 6.2.1 ***Brent*** *A partir de tout algorithme h^* admettant une performance relative de ρ pour le problème $\bar{P}|prec, c_{ij} = 1, p_i = 1|C_{max}$, nous*

pouvons obtenir un algorithme avec une performance relative $\rho^m \leq (1+\rho)$ *pour le problème* $P|prec, c_{ij} = 1, p_i = 1|C_{max}$ *(voir [45]).*

Preuve

Nous noterons par $C_{max}^{m,A}$, la longueur de l'ordonnancement donnée par l'heuristique proposé par l'algorithme 6.1 sur m processeurs et C_{max}^{∞,h^*} la longueur de l'ordonnancement donnée par l'heuristique h^* sur une infinité de processeurs. De manière similaire, nous noterons pas $C_{max}^{opt,m}$ (resp. $C_{max}^{opt,\infty}$) la longueur optimale sur m processeurs (resp. sur une infinité).

Etant donné que h^* est une ρ-approximation donc $C_{max}^{\infty,h^*} \leq \rho C_{max}^{opt,\infty}$ et nous avons trivialement $C_{max}^{opt,\infty} \leq C_{max}^{opt,m}$.

$$
\begin{aligned}
C_{max}^{m,A} &\leq \sum_{i=0}^{C_{max}^{\infty,h^*}-1} \left\lceil \frac{|X_i|}{m} \right\rceil \\
&\leq \sum_{i=0}^{C_{max}^{\infty,h^*}-1} \left(\left\lfloor \frac{|X_i|}{m} \right\rfloor + 1 \right) \\
&\leq \sum_{i=0}^{C_{max}^{\infty,h^*}-1} \left\lfloor \frac{|X_i|}{m} \right\rfloor + C_{max}^{\infty,h^*} \\
&\leq \sum_{i=0}^{C_{max}^{\infty,h^*}-1} \frac{|X_i|}{m} + C_{max}^{\infty,h^*} \\
&\leq C_{max}^{opt,m} + C_{max}^{\infty,h^*} \\
&\leq C_{max}^{opt,m} + \rho C_{max}^{opt,\infty} \\
&\leq C_{max}^{opt,m} + \rho C_{max}^{opt,m} \\
&\leq (1+\rho) C_{max}^{opt,m} \\
\rho^m &\leq (1+\rho)
\end{aligned}
$$

□

6.3 Etude de la topologie en anneau

6.3.1 Introduction

Nous avons décidé, dans un premier temps, d'étudier la topologie en anneau du fait de sa relative simplicité.

6.3.2 Complexité

Théorème 6.3.1 *Le problème de décider si une instance de* $(P, chain)|prec, c_{ij} = d(\pi^\ell, \pi^k), p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus trois est* $\mathcal{NP}$*-complet même dans le cas où le graphe de précédence est un graphe biparti de profondeur un.*

Preuve

Les preuves de ces deux théorèmes sont données dans [42].

□

6.3.3 Algorithmes d'approximation

Dans cette partie, nous présenterons deux algorithmes d'approximation avec garantie de performance : un en présence d'un nombre infini de processeurs et le second pour la version limité. Ces deux algorithmes sont basés sur l'algorithme classique d'approximation proposé par Munier et König [115] pour le problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$.

Pour ce problème, les auteurs proposent une solution basée sur une formulation par un programme linéaire en nombres entiers. Ils utilisent la procédure suivante : les contraintes d'intégrité sont relaxées et un ordonnancement réalisable est obtenu par une phase d'arrondie.

Pour le reste de la partie, nous donnons les notations suivantes pour l'ordonnancement σ^∞ :

- C^∞_{max} (resp. $C^{\infty,opt}_{max}$) désigne la longueur (resp. optimale) de σ^∞. X_i représente l'ensemble des tâches dont la fin d'exécution est à i, $i \geq 1$ dans σ^∞. Par définition, nous avons $\sum_i |X_i| = n$. Soit $m_{opt} = \max_i |X_i|$ le nombre maximum de processeurs utilisés pour l'ordonnancement σ^∞.

Théorème 6.3.2 *Pour le problème* $(\bar{P}, chain)|prec;\ c_{ij} = d(\pi^\ell, \pi^k); p_i = 1|C_{max}$*, il existe un algorithme d'approximation avec un ratio* $\rho^{chain,\infty} \leq \sqrt{n}$*, avec* n *le nombre de tâches et la borne supérieure est quasi-atteinte.*

Preuve

Pour problème $(\bar{P}, chain)|prec;\ c_{ij} = d(\pi^\ell, \pi^k); p_i = 1|C_{max}$, $C_{max}^{chain,\infty,opt}$ désigne la longueur optimale de l'ordonnancement.

Dans un premier temps, nous allons considérer deux cas simples.

- Si $C_{max}^{chain,\infty,opt} = 1$ alors il est facile de déterminer un ordonnancement optimal.
- Si $C_{max}^{chain,\infty,opt} = 2$ alors $C_{max}^{\infty,opt} = 2$. Sachant que l'algorithme proposé dans [115] est $\frac{4}{3}$-approché, celui-ci peut être utilisé pour déterminer la solution optimale.

Maintenant, nous étudions le cas où $C_{max}^{chain,\infty,opt} \geq 3$. Pour le problème $(\bar{P}, chain)|prec;\ c_{ij} = d(\pi^\ell, \pi^k); p_i = 1|C_{max}$, nous construisons un ordonnancement $\sigma^{chain,\infty}$ à partir de σ^∞. Soit $C_{max}^{chain,\infty}$ la longueur de l'ordonnancement $\sigma^{chain,\infty}$.

La construction $\sigma^{chain,\infty}$ dépend de m' (le nombre maximum de processeurs utilisé par σ^∞).

1. Si $m' \leq \sqrt{n}$ alors nous gardons l'ordonnancement σ^∞ et nous ajoutons $m'-2$ temps d'inactivité entre les X_i-tâches et les X_{i+1}-tâches avec i impair (voir la figure 6.2 pour une illustration). L'ordonnancement obtenu respecte toutes les contraintes du problème $(\bar{P}, chain)|prec;\ c_{ij} = d(\pi^\ell, \pi^k); p_i = 1|C_{max}$ sachant qu'il existe dans σ^∞, au plus un couplage entre les X_i et X_{i+1}-tâches. De plus entre X_i-tâches et les X_{i+2}-tâches l'écart entre les dates de début d'exécution est d'au moins de $m'-1$ (la distance maximum entre chaque paire de processeurs). $C_{max}^{chain,\infty}$ (resp. C_{max}^{∞}) désigne la longueur de $\sigma^{chain,\infty}$ (resp. σ^∞).

$$\begin{aligned} C_{max}^{chain,\infty} &\leq C_{max}^{\infty} + \frac{(m'-2)}{2} C_{max}^{\infty} \\ &\leq \frac{m'}{2} C_{max}^{\infty} \\ &\leq \frac{2m'}{3} C_{max}^{\infty,opt} \\ &\leq \frac{2m'}{3} C_{max}^{chain,\infty,opt} \\ &\leq \sqrt{n} C_{max}^{chain,\infty,opt} \end{aligned}$$

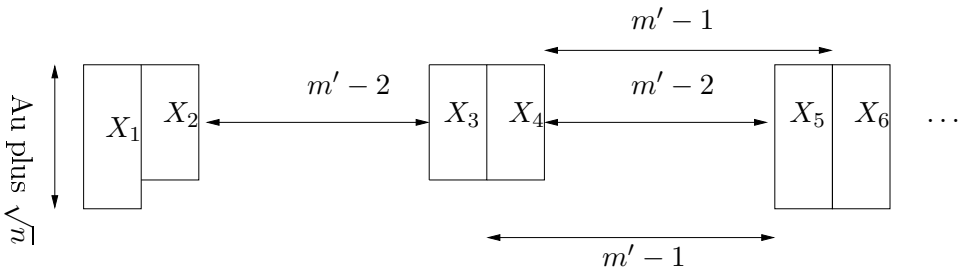


FIG. 6.2 – *Ordonnancement réalisable pour $\sigma^{chain,\infty}$ pour le cas où $m' \leq \sqrt{n}$*

2. Dans le cas contraire ($m' > \sqrt{n}$), $\forall X_i$, toutes les tâches sont exécutées en $\max(\lceil \frac{|X_i|}{\sqrt{n}} \rceil, 1)$ unités de temps $\sigma^{chain,\infty}$. Comme décrit précédent, nous ajoutons $\sqrt{n}-2$ unités de temps entre les X_i-tâches et les X_{i+1}-tâches pour i pair. De plus, soient deux ensembles de tâches X_{i_1} et X_{i_1+1} avec $i_1 \geq 1$ impair. Soit p la taille d'un matching entre les X_{i_1}-tâches et les X_{i_1+1}-tâches. Alors, nous avons $|X_{i_1}| \geq p$ et $|X_{i_1+1}| \geq p$. Nous exécutons les tâches couplées sur un même processeur, et les autres tâches au plus tôt.

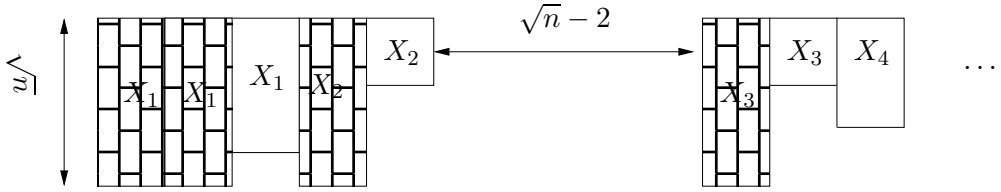


FIG. 6.3 – *Ordonnancement réalisable pour $\sigma^{chain,\infty}$ pour le cas $m' > \sqrt{n}$*

$$\begin{aligned} C_{max}^{chain,\infty} &\leq \sum_{i=1}^{n} \lceil \frac{|X_i|}{\sqrt{n}} \rceil + C_{max}^{\infty} + \frac{(\sqrt{n}-2)}{2} C_{max}^{\infty} \\ &\leq \sqrt{n} + C_{max}^{\infty} + \frac{(\sqrt{n}-2)}{2} C_{max}^{\infty} \\ &\leq \sqrt{n} + \frac{2\sqrt{n}}{3} C_{max}^{\infty,opt} \end{aligned}$$

Sachant $3 \leq C_{max}^{chain,\infty,opt}$ (par hypothèse) nous avons,

$$\begin{aligned} C_{max}^{chain,\infty} &\leq \frac{\sqrt{n}}{3} C_{max}^{chain,\infty,opt} + \frac{2\sqrt{n}}{3} C_{max}^{chain,\infty,opt} \\ &\leq \sqrt{n} C_{max}^{chain,\infty,opt} \end{aligned}$$

La borne supérieure est quasi-atteinte

Nous présentons une instance pour laquelle le ratio est strictement inférieur $\frac{2\sqrt{n}}{3}$.

Soit x un entier et soit le graphe de précendence $G' = (V', E')$ comme suit : $V' = \{u_i, v_i, s_i, u_{x+1} : 1 \leq i \leq x\}$ et $E' = \{(u_i \rightarrow v_i), (u_i \rightarrow s_i), (v_i \rightarrow u_{i+1}), (s_i \rightarrow u_{i+1}) : 1 \leq i \leq x\}$.

Pour le problème $(\bar{P}, chain)|prec; c_{ij} = d(\pi^\ell, \pi^k); p_i = 1|C_{max}$, l'ordonnancement optimal consiste à exécuter toutes les tâches sur le même $C_{max}^{chain,\infty,opt} = 3x + 1$.

Pour le problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$, l'algorithme Munier-König retourne un ordonnancement réalisable pour lequel $1 \leq i \leq x+1$, u_i est exécuté à l'instant $4(i-1)+1$. Pour chaque i, $1 \leq i \leq x$, v_i et s_i sont exécutées à l'instant $4(i-1)+3$. Cet ordonnancement utilise deux 2 processeurs. Ainsi, $C_{max}^{\infty} = 4x + 1$.

Nous considérons une instance où le graphe précédence est $\frac{\sqrt{n}}{2}$ copies du graphe G' où $x = 2\sqrt{n}$ (Voir Figure 6.4). En utilisant les mêmes arguments que précédemment, l'ordonnancement utilisent $\sqrt{n}$ processeurs. Nous obtenons $C_{max}^{chain,\infty,opt} = 6\sqrt{n}+1$ et $C_{max}^{\infty} = 8\sqrt{n}+1$. Nous obtenons au final pour notre algorithme $C_{max}^{chain,\infty} = 4n + \frac{\sqrt{n}}{2}$.

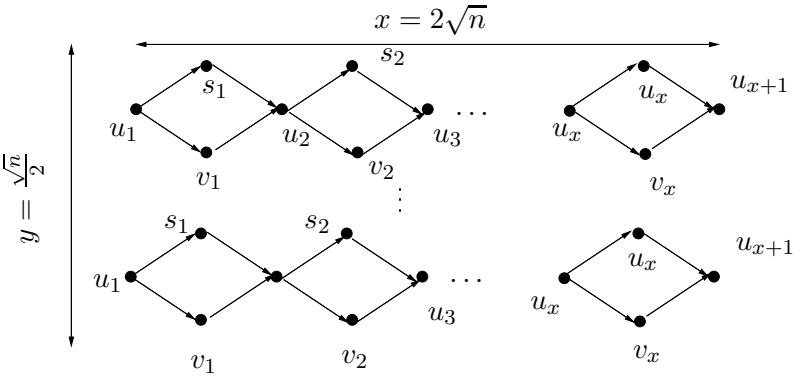


FIG. 6.4 – *Instance pour laquelle le ratio est strictement inférieur à* $\frac{2\sqrt{n}}{3}$

Ainsi, nous obtenu, $\rho^{chain,\infty} \leq \sqrt{n}$.

□

Maintenant, nous considérons le problème de processeurs est limité.

Théorème 6.3.3 *Pour le problème* $(P, chain)|prec;\ c_{ij} = d(\pi^\ell, \pi^k); p_i = 1|C_{max}$*, il existe un algorithme d'approximation avec pour un ratio* $(1+\frac{2m}{3})$*, où* m *est le nombre de processeurs et la borne supérieure est atteinte.*

Preuve

Nous utilisons le même algorithme que celui utilisé dans la preuve de théorème 6.3.2 cas (2) en remplaçant $\sqrt{n}$ par m. Ainsi nous obtenons

$$C_{max}^{chain} \leq \sum_{i=1}^{n} \frac{|X_i|}{m} + C_{max}^{\infty} + \frac{(m-2)}{2} C_{max}^{\infty}$$

Sachant $\sum_{i=1}^{n} \frac{|X_i|}{m} \leq C_{max}^{\infty,opt}$ et $C_{max}^{\infty} \leq \frac{4}{3} C_{max}^{chain,opt}$ (à partir [115]) , nous avons :

$$\begin{aligned} C_{max}^{chain} &\leq C_{max}^{\infty,opt} + \frac{4}{3} \times \frac{m}{2} C_{max}^{\infty,opt} \\ &\leq C_{max}^{chain,opt} + \frac{4}{3} \times \frac{m}{2} C_{max}^{chain,opt} \\ &\leq (1 + \frac{2}{3}m) C_{max}^{chain,opt} \end{aligned}$$

Borne supérieure atteinte : Nous allons construire une instance pour laquelle le ratio est $1+\frac{2m}{3}$. Nous considérons le graphe G ayant $(\frac{m}{2}+1)$ copies du graphe G' avec $x = \frac{1}{3}(\frac{2n}{m+2} - 1)$ décrit dasb la preuve du Théorème 6.3.2. Alors en utilisant les mêmes arguments que précédemment nous obtenons $C_{max}^{chain,\infty,opt} = \frac{2n}{m+2}$ et $C_{max}^{\infty} = \frac{4}{3}(\frac{2n}{m+2} - 1) + 1$. Et donc $C_{max}^{chain,m} = (2m + 3)x + 1$.

□

6.4 Etude de la topologie en étoile

6.4.1 Introduction

Dans une second temps, nous poursuivons notre étude sur la topologie en étoile. A la différence de l'anneau le diamètre de l'étoile est constant.

Nous notons une étude similaire sur une architecture du type étoile : dans [36] les auteurs considèrent le problème de maître-esclave sur une plate-forme hétérogène. Ils considère le problème d'allouer un ensemble de tâches (un très grand nombre) indépendantes de taille équivalentes sur une plate-forme de calculs hétérogènes. Au lieu de considérer un critère classique, ils proposent un algorithme polynomial basé sur une formulation par un programme linéaire dans le but d'obtenir un ordonnancement optimal en régime permanent. Il est important de noter que notre problème est proche du problème d'ordonnancement maître-esclave.

6.4.2 Complexité

Le challenge pour le problème $(P, étoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1|C_{max}$, est de le classifier au sens de la complexité. Nous montrerons que ce problème n'admet pas d'algorithme approché avec une performance garantie inférieure à $\frac{6}{5}$ pour la minimisation de la longueur de l'ordonnancement. Nous proposons également pour $C_{max} = 4$ un algorithme polynomial. Les résultats présentés dans cette partie sont issus de [169].

Dans la suite, nous allons montré la $\mathcal{NP}$-complétude pour $C_{max} = 5$. La preuve du Théorème 6.4.1 est basée sur la preuve proposée dans [98] et [89].

Théorème 6.4.1 *Le problème de décider si une instance de* $(P, étoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus cinq est* $\mathcal{NP}$*-complet.*

Preuve

Notre preuve est basée sur une réduction à partir de *Maximum Clique* (voir [67], section 2.5).

Soit Φ' une instance de *Maximum Clique*, nous construisons une instance Φ pour $(P, étoile) \mid prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1 \mid C_{max}$.

Soit $l = \frac{k(k-1)}{2}$ le nombre d'arêtes de la clique de taille k.

$\forall v \in V$ nous créons quatre tâches T_v, K_v, M_v et N_v et $\forall e \in E$ nous créons une tâche L_e. Cette tâche définit les ensembles T, K et L sur lequel nous ajoutons les contraintes suivantes :

- $T_v \rightarrow K_v \rightarrow M_v \rightarrow N_v$,
- $T_{v_1} \rightarrow L_e$ et $T_{v_2} \rightarrow L_e$ si $e = (v_1, v_2) \in E$.

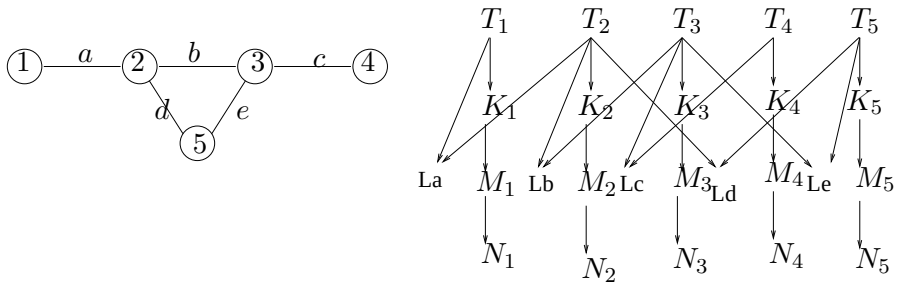


FIG. 6.5 – *Illustration de l'instance construite pour le problème d'ordonnancement à partir du problème $\mathcal{NP}$-complet clique*

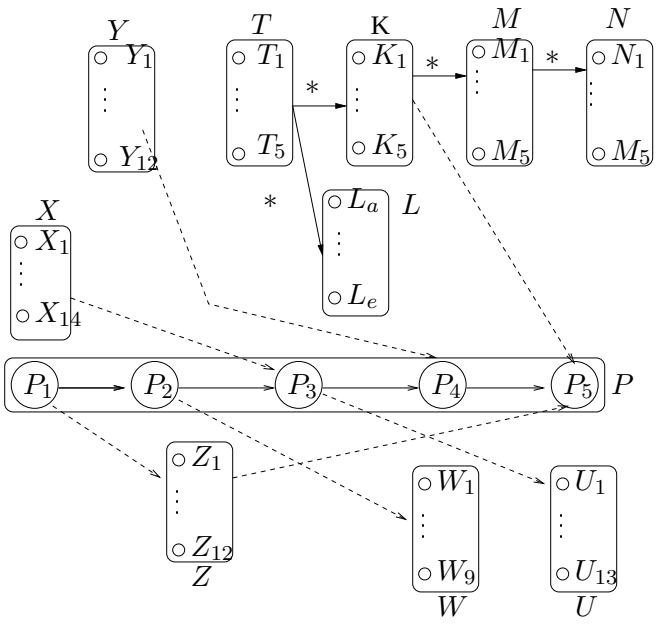
Nous créons également six ensembles de tâches « dummy » :

- $P = \{P_1, P_2, P_3, P_4, P_5\}$
- $X = \{X_1, \ldots, X_{m-k-1}\}$
- $Y = \{Y_1, \ldots, Y_{m-|V|-1}\}$
- $Z = \{Z_1, \ldots, Z_{m-|V|-1}\}$
- $W = \{W_1, \ldots, W_{m-l-1-|V|}\}$
- $U = \{U_1, \ldots, U_{m-|E|+k-1+l-|V|}\}$,

avec pour contraintes de précédence :

- $\forall x \in X : x \rightarrow P_3$
- $\forall y \in Y : y \rightarrow P_4$
- $\forall z \in Z : P_1 \rightarrow z$ and $z \rightarrow P_5$
- $\forall w \in W : P_2 \rightarrow w$
- $\forall u \in U : P_3 \rightarrow u$
- $\forall k \in K : k \rightarrow P_5$
- $P_i \rightarrow P_{i+1}$ for $1 \leq i \leq 4$

le nombre de processeurs est $m = 2\max\{|V|+1, l+1+|V|, |E|-k+1-l+|V|\}$.



$*$ Contraintes de précédence entre les ensembles T, K, M, N et L sont données par la figure 6.5

FIG. 6.6 – *Les contraintes de précédence entre les ensembles X, Y, Z, W et U à partir de l'instance donné par la figure 6.5*

• Nous supposons qu'il existe une clique de taille k dans le graphe G, et nous allons montrer qu'il existe un ordonnancement de longueur cinq. Pour cela il suffit de construire d'ordonnancer les tâches comme la figure 6.7 le propose.

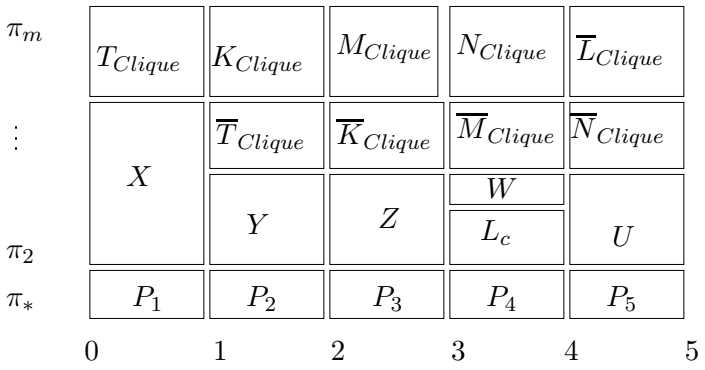


FIG. 6.7 – *Ordonnancement de longueur cinq*

Notation : T_{clique} (resp. $\bar{T}_{clique}$) désigne l'ensemble des tâches qui sont (resp. qui ne sont pas) éléments de la clique de taille k.

• Réciproquement, nous supposons qu'il existe un ordonnancement de longueur cinq. Nous allons prouver qu'il existe une clique de taille k dans le graphe G.

Nous allons faire quelques remarques essentielles sur un ordonnancement réalisable de longueur cinq.

1. Les tâches de P doivent être allouées sur le processeur π^*. D'une part les tâches de P forme un chemin de longueur cinq. Elles doivent être exécutées consécutivement sur le même processeur. D'autre part, il existe un chemin $c = \{P_1, z, P_5\}$ $(\forall z \in Z)$. Ainsi deux il existe deux délais de communication ($P_1 \rightarrow z$ et $z \rightarrow P_5$). Si un tâche de P_i n'est pas affectée de manière consécutive sur le processeur π^* le début d'exécution de P_5 est au moins de $t = 5$.
2. Chaque tâche de X (resp. U) est prédécesseur (resp. successeur) de P_3 alors toutes les tâches de X (resp. U) doivent être ordonnancées à $t = 0$ (resp. $t = 4$).
3. Chaque tâche de Z est successeur de P_1 et prédécesseur de P_5 alors les tâches de Z doivent être exécutées à $t = 2$.
4. Chaque tâche de Y est prédécesseur de P_4 alors les tâches de l'ensemble Y doivent être ordonnancées à $t = 0$ ou $t = 1$.
5. Chaque tâche de W sont successeurs de P_2 alors les tâches de W doivent être exécutées à $t = 3$ où à $t = 4$.

Remarquons qu'après l'ordonnancement des ensembles P, X, Y, Z, W et

U, nous avons $4|V|+|E|$ processeurs inactifs pour $4|V|+|E|$ tâches à exécuter venant des ensembles T, K et L. Nous allons étudier ces trois ensembles :

1. Tous les chemins de longueur quatre doivent être exécutés consécutivement sur le même processeur. Ainsi, pour une tâche $v \in V$ alors le chemin $T_v \to K_v \to M_v \to N_v$ admet un début d'exécution soit à $t = 0$ ou $t = 1$.
2. Les tâches de l'ensemble K sont exécutées à $t = 2$ ou avant sachant que les tâches de K sont des prédécesseurs de P_5. Chaque tâche de K est également un successeur d'une tâche de T. Sachant qu'il n'existe aucun temps d'inactivité sur le processeur π^*, tous les délais de communication admettent un coût de deux. Alors, les tâches de K doivent être ordonnancées sur le même processeurs que son prédécesseur de T. Ainsi, toutes les tâches de K ne peuvent qu'être affectées à $t = 1$ ou à $t = 2$.
3. Les tâches de l'ensemble L possèdent deux prédécesseurs dans T. Comme toutes les tâches de T sont ordonnancées sur des processeurs différents, il existe un délai de communication entre les tâches de L et leurs prédécesseurs dans T. Soit L_e une tâche de L. Si deux prédécesseurs de L_e sont exécutées à $t = 0$ alors L_e peut être affectée à l'instant $t = 3$, sinon L_e doit être exécutées à $t = 4$.
4. A $t = 0$, il existe au plus k instants d'inactivité et à $t = 3$ il existe au plus l instant d'inactivité. Supposons que $h < k$ tâches de T sont affectées à $t = 0$, alors un graphe ayant $h < k$ sommets ne peut avoir plus de l arêtes. Et donc, à $t = 3$ il y a au plus de l tâches qui peuvent être exécutées. Comme le nombre de tâches est égal au nombre de slots, nous ne pouvons avoir un ordonnancement réalisable avant l'instant $t = 5$.

   Supposons maintenant que nous avons k tâches ordonnancées à $t = 0$. La seule manière d'obtenir l tâches exécutées à $t = 3$ c'est quand le sous-graphe défini par les k tâches forme une clique. A contrario, si nous avons moins de l tâches à $t = 3$, nous aurions des temps d'inactivité, ce qui est impossible dans tout ordonnancement réalisable de longueur cinq.

Ainsi, nous avons montré que dans G il existe une clique de taille k. Ceci conclut la preuve du Théorème 6.4.1.

□

Corollaire 6.4.1 *Il n'existe pas d'algorithme approché avec une performance relative inférieure à* $\frac{6}{5}$ *pour le problème* $(P, étoile) \mid prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1 \mid C_{max}$

Corollaire 6.4.2 *Il n'existe pas d'algorithme approché avec une performance relative inférieure à* $\frac{6}{5}$ *pour le problème* $(P, étoile) \mid prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1, dup \mid C_{max}$

Preuve

La preuve vient directement de la preuve du Théorème 6.4.1. En effet, à partir de la construction de l'instance du problème $(P, étoile)|, prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1, |C_{max}$, nous savons qu'aucune tâche ne peut être dupliquée. □

6.4.3 Un algorithme polynomial

Théorème 6.4.2 *Le problème de décider si une instance de* $(P, étoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1|C_{max}$ *possède un ordonnancement de longueur quatre est polynomial.*

Preuve

Nous proposons une idée de la preuve. Le graphe de précédence ne peut admettre plus que $4m$ tâches. Il existe donc au plus $(4m)^4$ allocations possibles sur le processeurs π^*. A partir d'une affectation réalisable sur le processeur π^*, il est nécessaire d'exécuter les autres tâches sur les processeurs $\pi^i, 1 \leq i \leq n$. Cependant, les délais de communications entre deux processeurs π^i et $\pi^{i'}$ avec $i \neq i'$ et $i, i' \in \{1, \ldots, m\}$ est égal à deux. Sachant que pour le problème $P|prec; p_i = 1; c_{ij} = 2|C_{max} \leq 4$ (voir [24]), les auteurs ont développés un algorithme polynomial. Ainsi nous pouvons adapter leur preuve à notre problème.

Remarque : L'affectation des quatre tâches sur le processeur π^* induit quleques contraintes sur les dates de début d'exécution des autres tâches, mais la preuve s'adapte aisément.

□

6.4.4 Minimisation de la somme des temps de complétude

Dans cette section, nous allons étendre le résultat de non-approximabilité sur l'étoile pour la minimisation de la longueur de l'ordonnancement au problème de la minimisation de la somme des temps de complétude.

Théorème 6.4.3 *Il n'existe pas d'algorithme approché avec une performance relative inférieure à* $(1 + \frac{1}{13})$ *pour le problème* $(P, étoile)\ |prec; c_{ij} = d(\pi^l, \pi^h); p_i = 1| \sum_j C_j$.

Preuve La preuve du Corollaire 6.4.3 est une conséquence immédiate du Théorème de l'Impossibilité, (voir [50], [67], section 2.2.2).

□

6.4.5 Algorithme approché sur l'étoile

Dans la section 6.4.2, nous avons proposé un seuil d'approximation pour n'importe quel algorithme pour le problème $(P, étoiletoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1|C_{max}$. Maintenant, nous allons proposer un algorithme approché avec une performance relative non triviale.

6.4.5.1 Description de l'algorithme

L'algorithme approché utilisent trois étapes : à chaque étape nous appliquons un algorithme spécifique pour un problème d'ordonnancement donné ([115], [74], [165]). Dans les deux premières étapes un ordonnancement est produit (les deux ordonnancements produits ne sont pas des ordonnancement réalisable pour notre problème). Ce n'est qu'à la dernière étape que nous obtenons un ordonnancement réalisable pour notre problème $(P, étoile)|prec; c_{ij} = d(\pi^l, \pi^h); p_i = 1|C_{max}$.

Dans la première étape, un ordonnancement (noté par S) pour le problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$ est produit. Pour ce problème, Munier et König [115] propose un algorithme (4/3)-approché, qui est basé sur la relaxation d'un programme linéaire en nombres entiers, et un ordonnancement réalisable est produit par arrondis.

La seconde étape de notre algorithme produit un ordonnancement (noté S') à partir de S en utilisant le principe de la dilatation proposé dans [74], et dans la section 4.2.5 pour le problème $\bar{P}|prec; c_{ij} \geq 2; p_i = 1|C_{max}$. Cet algorithme est appelé $Expand()$.

La troisième étape produit un ordonnancement S'' sur une topologie étoile à partir de S' (réalisable pour le problème $(P, étoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1|C_{max}$) en utilisant le principe du pliage [165]. L'algorithme du pliage construit un ordonnancement sur un nombre fini de processeurs à partir d'un ordonnancement sur une infinité de processeurs. Cet algorithme sera noté $Folding$.

6.4.5.2 Algorithme et analyse de la performance relative

Algorithme 6.2 Ordonnancement pour le problème $(P, étoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1|C_{max}$

$G = (V, E)$ un graphe de précédence, m le nombre de processeur
S =Ordonnancement pour le problème $(P, étoile)|prec, c_{ij} = d(\pi^i, \pi^j), p_i = 1\ |C_{max}$
Soit S la solution pour $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$
$S' = Expand(S)$
$S'' = Folding(S')$

Théorème 6.4.4 *L'algorithme 6.2 donne une* 4*-approximation pour le problème* $P, étoile|prec; c_{ij} = d(\pi^l, \pi^h); p_i = 1|C_{\max}$.

Preuve

Maintenant, étudions la performance relative de l'algorithme. Dans un premier temps. En utilisant l'algorithme de Munier et König [115], nous avons :

$$C_{max}^{UET-UCT,\infty} \leq \frac{4}{3} C_{max}^{opt,UET-UCT,\infty}$$

Alors nous calculons $Schedule(UET - LCT(c = 2), \infty)$ en utilisant [74] ceci n'est pas une solution optimale pour le problème UET-LCT . Comme indiqué précédemment la nouvelle performance relative est $\frac{2(c+1)}{3}$. Alors nous avons

$$\begin{aligned} C_{max}^{UET-LCT(c=2),\infty} &\leq \frac{3}{2} \cdot \frac{4}{3} C_{max}^{opt,UET-LCT,\infty} \\ C_{max}^{UET-LCT(c=2),\infty} &\leq 2 C_{max}^{opt,UET-LCT(c=2),\infty} \end{aligned}$$

Maintenant nous avons un ordonnancement sur le problème UET-LCT avec des délais de communication égaux à 2 et sur une infinité de processeurs. Cet ordonnancement peut-être utilisé pour une topologie en étoile ayant un nombre infini de processeurs (un seul maître).

Lemme 6.4.1 *L'écart entre l'ordonnancement sur un topologie complète de processeurs et un graphe admettant un diamètre de longueur* d*, pour des tâches unitaires est* $2(d + 1)/(d + 2)$.

Preuve Soit $p = \{x_1, x_2, \ldots, x_n\}$ un chemin critique (chemin qui donne la longueur de l'ordonnancement). Supposons qu'il existe un délai de communication entre chaque paire de tâches (x_i, x_{i+1}) avec $1 \leq i < n$. Pour le système UET-LCT (avec des délais de communication égaux à d) la longueur de l'ordonnancement serait de $(1+d)n - d$. Mais chaque délai de communication est unitaire alors la longueur de l'ordonnancement est $2n+1$. Dans une topologie en étoile, le cas où les communications sont unitaires apparaît quand les communications interviennent entre le processeur maître et les satellites. □

Alors, nous obtenons, $C_{max}^{opt,UET-LCT(c=2),\infty} \leq \frac{3}{2} C_{max}^{opt,étoile,\infty}$, et donc :

$$\begin{array}{rcl} C_{max}^{étoile,\infty} \leq C_{max}^{UET-LCT(c=2),\infty} & \leq & 2C_{max}^{opt,UET-LCT(c=2),\infty} \\ C_{max}^{étoile,\infty} & \leq & 3C_{max}^{opt,étoile,\infty} \end{array}$$

Maintenant, nous transformons cet ordonnancement en utilisant une infinité de processeurs en un ordonnancement sur un nombre fini. Ceci est facile en utilisant la méthode proposée dans [165], voir 6.2.2. La valeur du ratio augmente d'une seule unité :

$$C_{max}^{étoile} \leq (3+1) C_{max}^{opt,étoile} \leq 4 C_{max}^{opt,étoile}$$

□

Tous ces résultats peuvent étendus au cas où le graphe G^* admet un diamètre de $d \geq 1$, et G^* formant une étoile.

6.4.6 Etude de la topologie d'un graphe structuré en étoile de diamètre d

Dans cette partie, nous allons généraliser tous les résultats de complexité et d'approximation. Nous considérons une étoile avec un processeur maître π^* tel que $c_{ij} = \theta$ et $d(\pi^l, \pi^*) = 1, , \ \forall i \in \{1, \ldots, m-1\}$.

Théorème 6.4.5 *Le problème de décider si une instance de* $(P, étoile)|prec, c_{ij} = \theta; d(\pi^*, \pi^j) = 1, p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus* $(2\theta + 3)$ *est* $\mathcal{NP}$*-complet.*

Preuve

La preuve est similaire à la preuve du Théorème 6.4.1. Nous proposons une transformation polynomiale $clique \propto (P, étoile)|prec, c_{ij} = \theta; d(\pi^*, \pi^j) = 1, p_i = 1|C_{max} = 2\theta + 3$. Nous donnerons simplement la transformation polynomiale.

- $\forall v \in V$ une tâche T_v et un chemin de longueur $2\theta - 1$ ($K_v = \{K_v^1, K_v^2, \ldots, K_v^{2\theta-1}\}$ sont créés. Nous introduisons les ensembles T et $K = \sum_{v \in V} K_v$. Nous ajoutons les contraintes suivantes : $T_v \rightarrow K_v^1 \rightarrow K_v^2 \rightarrow \ldots \rightarrow K_v^{2\theta-1} \rightarrow M_v \rightarrow N_v$.
- $\forall e \in E$, une tâche L_e est introduite. Nous définissons l'ensemble, par la même, l'ensemble des tâches L. Nous ajoutons les contraintes de précédence suivants :
  - $T_v \rightarrow K_v^1 \ldots K_v^{2\theta-1} \rightarrow M_v \rightarrow N_v$,
  - $T_{v_1} \rightarrow L_e$ et $T_{v_2} \rightarrow L_e$ if $e = (v_1, v_2) \in E$.
- Nous créons un chemin de longueur $2\theta + 3, P = \{P_1, \ldots, P_{2\theta+3}\}$ et un ensemble de tâches « dummies » :
  - $X = \{X_1, \ldots, X_{m-k-1}\}$,
  - $W = \{W_1, \ldots, W_{m-1-(|E|-l)-|V|}\}$
  - $U = \{U_1, \ldots, U_{m-|E|+k-1+l-|V|}\}$
  - and $Z = \{Z_k^i\}$ avec $i \in \{1, \ldots, 2\theta - 1\}$ et $k \in \{1, \ldots, m - 1 - |V|\}$ avec pour contraintes de précédence : $Z_k^1 \rightarrow Z_k^2 \rightarrow \ldots Z_k^{2\theta-1}$, $\forall k \in \{1, \ldots, m - 1 - |V|\}$.
- Nous ajoutons les contraintes de précédence entre les ensembles de tâches :
  - $\forall x \in X, x \rightarrow P_{\theta+2}$;
  - $P_{\theta+2} \rightarrow u, \forall u \in U$;
  - $P_{\theta+1} \rightarrow w, \forall w \in W$;
  - $Z_k^1 \rightarrow P_{\theta+3}, \forall k \in \{1, \ldots, m - 1 - |V|\}$;
  - $P_\theta \rightarrow Z^{2\theta-1}$, $\forall k \in \{1, \ldots, m - 1 - |V|\}$;
  - $\forall k \in K, K_1^\theta \rightarrow P_{2\theta+3}$;
  - $\forall l \in L, P_{\theta+1} \rightarrow l$.

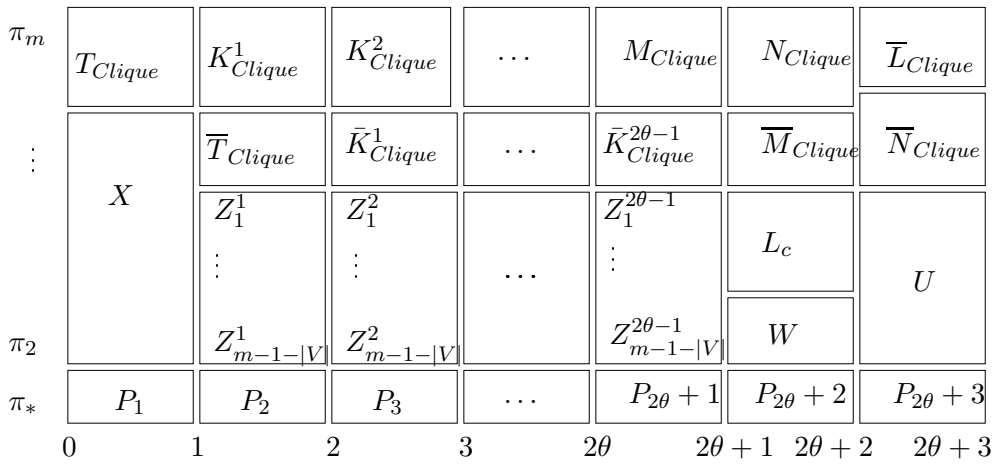


Fig. 6.8 – *Un ordonnancement de longueur* $2\theta + 3$

En utilisant les mêmes arguments que précédemment (voir le Théorème 4.2.3), nous pouvons conclure que le problème de décider si une instance de $(P, étoile)|prec, c_{ij} = \theta; d(\pi^*, \pi^j) = 1, p_i = 1|C_{max}$ possède un ordonnancement de longueur au plus $(2\theta + 3)$ est $\mathcal{NP}$-complet. □

Corollaire 6.4.3 *Il n'existe pas d'algorithme approché avec une performance relative strictement inférieure à* $1 + \frac{1}{2\theta+3}$ *(à moins que* $\mathcal{P} = \mathcal{NP}$*) pour le problème* $(P, étoile)|prec, c_{ij} = \theta; ; d(\pi^*, \pi^j) = 1; p_i = 1|C_{max}$.

Corollaire 6.4.4 *Il n'existe pas d'algorithme approché avec une performance relative strictement inférieure à* $1 + \frac{1}{2\theta+3}$ *(à moins que* $\mathcal{P} = \mathcal{NP}$*) pour le problème* $(P, étoile)|prec, c_{ij} = \theta; d(\pi^*, \pi^j) = 1, p_i = 1, dup|C_{max}$.

Corollaire 6.4.5 *Il n'existe pas d'algorithme approché avec une performance relative strictement inférieure à* $1 + \frac{1}{11+2\theta}$ *(à moins que* $\mathcal{P} = \mathcal{NP}$*) pour le problème* $(P, étoile)|prec, c_{ij} = \theta; d(\pi^*, \pi^j) = 1, p_i = 1| \sum C_j$.

Théorème 6.4.6 *L'algorithme proposé dans 6.4.5, conduit à un algorithme approché avec une performance relative de* $(\frac{4}{3}\frac{(2\theta+1)^2}{2\theta+2} + 1)$ *pour le problème* $(P, étoile)|prec; c_{ij} = \theta; d(\pi^*, \pi^h) = 1; p_i = 1|C_{\max}$.

Preuve

La valeur du 4/3 provient de l'algorithme de Munier and König [115]. Le ratio est multiplié, une première fois, par $(2\theta + 1)/2)$, cette valeur est issue du processus de dilatation [74], et $2\frac{(2\theta+1)}{2\theta+2}$ résulte de l'application du Lemme 6.4.1. La valeur additionnelle provient du Théorème 6.2.1 (ou voir [45]).

Remarque 6.4.1 *L'ordre des opérations admt un impact sur la valeur de la performance relative. En effet, le principle du pliage peut-être utilisé sur l'ordonnancement obtenue par l'algorithme de Munier and König [115]. Alors, nous obtenons un ordonnancement sur m processors. Ensuite, nous procédons au principe de la dilatation. Ainsi, cet ordre sur les opérations conduit à un algorithme approché avec une performance relative de $\frac{7}{4} \times \frac{(d+1)^2}{4}$. Ainsi, si $\theta = 2$ nous obtenons le ratio $\frac{21}{4}$.*

□

6.5 Etude de la topologie d'un graphe structuré de diamètre d

6.5.1 Introduction

Dans les sections précédentes, nous avons obtenu plusieurs seuils d'approximation pour diverses topologies. Mis à part la topologie en étoile, les autres topologies admettent un seuil d'approximation de 4/3. La question qui est légitime de se poser : « Peut-on exhiber une classe de graphes pour laquelle nous sommes capables de produire, via le théorème de l'Impossibilité, un seuil d'approximation de 4/3 ? ».

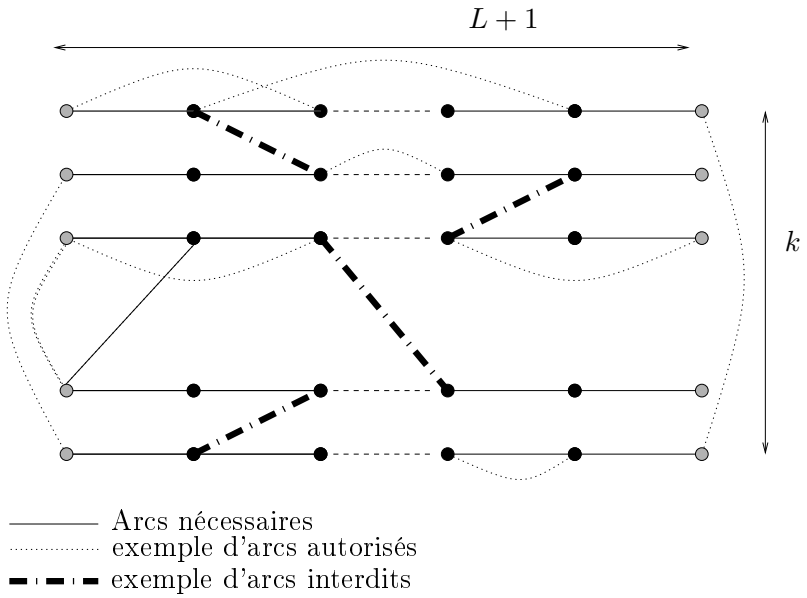
La seconde question consiste à développer un algorithme d'approximation générique pour toute topologie représentée par un graphe de diamètre d.

6.5.2 Complexité

Nous allons définir une classe de graphes appelé la classe Prisme de la manière suivante :

Définition 6.5.1 *Un prisme $P = (V_P, E_P)$ de taille k et de longueur L $(k, L \in \mathbb{N})$ est un graphe non orienté pour lequel*

- *deux ensembles de sommets K et K' avec $K \subset V_P$, $K' \subset V_P \backslash \{K\}$, et $|K| = |K'| = k$. Les sommets sont nommés $s_1, \ldots, s_k$ (resp. $s'_1, \ldots, s'_k$).*



- sommet qui n'appartiennent pas $K \cup K'$
- sommets dans K où dans K'

FIG. 6.9 – Un exemple d'un Prisme graphe de taille k et de longueur L

- *il existe un ordre sur* K *et* K' *tel que* $\forall s_i \in K, s'_i \in K,\ 1 \leq i \leq k$*) il existe un chemin de longueur* L *noté par* C_i *entre* s_i *et* s'_i
- $(i \neq j) \wedge x \in C_i \backslash \{s_i, s'_i\} \wedge y \in C_j \backslash \{s_j, s'_j\} \Rightarrow (x, y) \notin E_P$

De plus, la taille du primse est polynomiale en k *car polynomial en* L*. Une illustration est donnée par la figure 6.9.*

Définition 6.5.2 *Soit* $\mathcal{G}$ *une famille de graphe.* $\mathcal{G}$ *possède la propriété Prisme si et seulement si* $\forall n_0, \forall n_1 \in \mathbb{N}\ \exists G \in \mathcal{G}$*,* G *contient* ***un unique*** *sous-graphe* $G_1 = (V_1, E_1)$ *de* G *induit par les sommets* $V_1 \subset V$ *qui est un prisme de taille* $k = n_0$ *et de longueur* $L = n_1$*.*

Lemme 6.5.1 *La classe de graphe* $\mathcal{G}$ *est non vide.*

Preuve

Nous avons montré dans [43] que cette classe de graphe est non vide. Pour plus de détail, voir la partie discussion en 6.5.3.

□

Théorème 6.5.1 *Le problème de décider si une instance de* $(P, G^*)|prec; c_{ij} = d(\pi^k, \pi^l) \leq d; p_i = 1|C_{max}$ *possède un ordonnancement de longueur au plus trois est* $\mathcal{NP}$*-complet avec* $G^* \in \mathcal{G}$.

Preuve

Il est facile de voir que $\{P, G^*\}|prec, c_{ij} = d(\pi^k, \pi^l) \leq d, p_i = 1|C_{max} \leq 3 \in \mathcal{NP}$.

Considérons une instance $\mathcal{I}$ de n-PARTITION, $(\mathcal{A} = \{a_1, \ldots, a_{3n}\}, B)$, où $\sum_{a \in \mathcal{A}} s(a) = nB$. Sans perte de généralité, nous supposons que $\forall a \in \mathcal{A}, s(a) \geq 2$.

Considérons une instance $\mathcal{I}$ du problème 3-PARTITION, nous construisons l'instance $\mathcal{I}'$ du problème d'ordonnancement $(P, G^*)|prec; c_{ij} = d(\pi^\ell, \pi^k) \leq d; p_i = 1|C_{max} \leq 3$ avec $G^* \in \mathcal{G}$ de la manière suivante :

Le graphe de précédence $\tilde{G}$ est composé de deux graphes disjoints, notés par $\mathcal{W}$ et $\mathcal{Z}$ où le graphe $\mathcal{Z}$ est une collection des graphes $Z^{s(a_j)}$ i.e. $\mathcal{Z} = \cup_{a_j \in \mathcal{A}} Z^{s(a_j)}$.

Le graphe $\mathcal{Z}$, qui est une collection des graphes Z^i défini de la manière suivante :

Soit $i > 1$ un entier. Le graphe Z^i est constitué de $4 \times i$ sommets $Z^i[k, 0]$, $Z^i[k, 1]$ avec $0 \leq k < 2i$. Les contraintes de précédence entre ces tâches sont les suivantes :

- $Z^i[j, 0] \rightarrow Z^i[j, 1]$ $\forall j, 0 \leq j \leq 2i - 1$,
- $Z^i[2j, 0] \rightarrow Z^i[2j + 1, 1]$ $\forall j, 0 \leq j \leq i - 1$,
- $Z^i[2j, 0] \rightarrow Z^i[2j - 1, 1]$ $\forall j, 1 \leq j \leq i - 1$.

Remarque 6.5.1 *Dans n'importe ordonnancement valide de longueur trois sur une chaîne de longueur de* $2i$*, les tâches du graphe* Z^i *sont exécutées de la manière suivante :*$\forall j, 0 \leq j \leq 2i - 1$,

- *les tâches* $Z^i[j, 0]$ *et* $Z^i[j, 1]$ *sont exécutées sur* π^j,
- *les tâches* $Z^i[j, \ell]$ *sont ordonnancées à l'instant* $\ell, \forall \ell \in \{0, 1\}$*, si* j *est pair,*
- *les tâches* $Z^i[j, \ell]$ *sont exécutées à* $\ell + 1$, $\forall \ell \in \{0, 1\}$*, sinon.*

Une illustration est donnée par la figure 6.10 pour le graphe Z^2 et la figure 6.11 décrit un ordonnancement valide comme indiqué par la remarque 6.5.1.

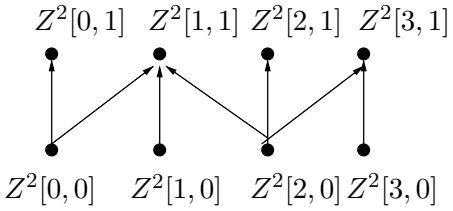


FIG. 6.10 – *Graphe Z^2*

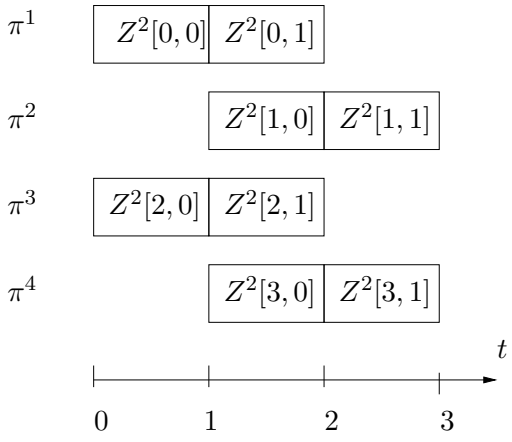


FIG. 6.11 – *Ordonnancement réalisable pour le graphe Z^2 en temps trois sur une chaîne de longueur quatre*

Remarque 6.5.2 *Un chemin de longueur l admet $l+1$ sommets.*

Nous construisons maintenant le graphe de précédence $\mathcal{W}$. Soit $G^* = (V^*, E^*)$ un graphe tel que $G^* \in \mathcal{G}$, $V^* = \{v_1^*, \ldots, v_{n^*}^*\}$. En se basant sur la définition 6.5.2, nous savons qu'il existe un unique sous-graphe $G = (V \subset V^*, E \subset E^*)$ de taille k et de longueur L ayant les propriétés requises. Par la suite nous posons que $k = n$ et $L = 2B + 1$ et la taille de $G^* = (V^*, E^*)$ est polynomial en k. Remarquons que $n^* >> 2B$.

Ainsi $V^* = V' \cup V$ est partitionné en deux sous-ensembles,

- avec $V = \{v_1^*, \ldots, v_{2n(B+1)}^*\}$ les sommets appartenant au graphe G, et donc constituent les sommets des n uniques chemins de longueur $(2B+1)$ respectant les caractéristiques de la définition 6.5.1.
- et $V' = \{v_{2n(B+1)+1}^*, \ldots, v_{n^*}^*\}$, l'ensemble des sommets qui n'appartiennent au sous-graphe G.

La caractérisation du graphe $\mathcal{W}$ est la suivante :

- $\forall i \in \{1, \ldots, 2n(B+1)\}$, nous créons un chemin de longueur deux avec trois tâches $v_i^*[0], v_i^*[1]$ et $v_i^*[2]$, avec les arcs $v_i^*[0] \rightarrow v_i^*[1] \rightarrow v_i^*[2]$. L'ensemble de ces tâches sera noté par $\mathcal{V}_1 = \{v_i^*[j] \backslash \forall i \in \{1, \ldots, 2n(B+1)\}, j\{0,1,2\}\}$. La taille de $\mathcal{V}_1$ est $6n(B+1)$.
- $\forall i \in \{2n(B+1)+1, \ldots, n^*\}$, nous créons un chemin de longueur trois $v_i^*[0] \rightarrow v_i^*[1] \rightarrow v_i^*[2]$. L'ensemble de ces tâches sera noté par $\mathcal{V}'$. Le nombre de ces tâches est $3(n^* - 2n(B+1))$ avec $n^* = |V^*|$.
- $(k,l) \in E^*$, nous ajoutons les arcs $v_k^*[0] \rightarrow v_l^*[2]$ et $v_l^*[0] \rightarrow v_k^*[2]$.

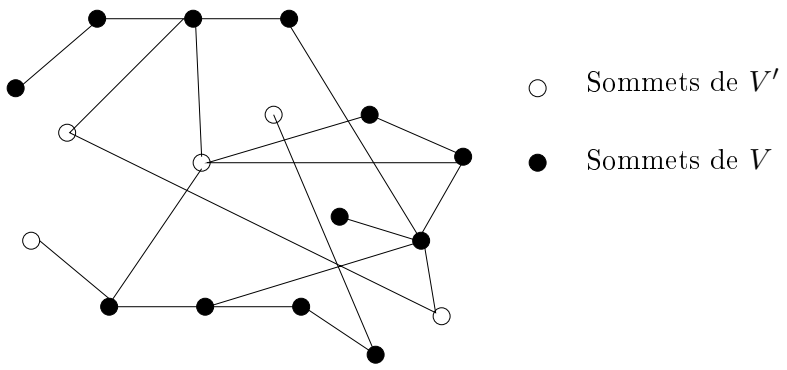
Maintenant, nous retirons du précédent graphe $4nB$ tâches[2].

Soient les ensembles suivants ;

- $J_0 = \{2i(B+1) | i = \{1, 2, \ldots, n\}\}$,
- $J_1 = \{2i(B+1)+1 | i \in \{0, 1, 2, \ldots, n-1\}$,
- $I_0 = \{k \in \{1, \ldots, 2n(B+1)\} \backslash \{J_0 \cup J_1\}$ et $|k$ est pair$\}$,
- $I_1 = \{k \in \{1, \ldots, 2n(B+1)\} \backslash \{J_0 \cup J_1\}$ et $|k$ est impair$\}$.

I_0 et I_1 sont deux ensembles d'index. Nous retirons de la construction les tâches $v_k^*[0]$, $v_k^*[1]$ avec $k \in I_0$, (resp. $v_k^*[1]$, $v_k^*[2]$ avec $k \in I_1$). Ces tâches sont retirées de l'ensemble des tâches $\mathcal{V}_1$. Nous notons par $\mathcal{K}$ cet ensemble de tâches supprimées.

Et nous avons $\mathcal{V} = \mathcal{V}_1 \backslash \mathcal{K}$ avec $|\mathcal{V}| = 2nB + 6n$.



Graphe G^*

FIG. 6.12 – *Début de l'illustration de la construction du graphe $\mathcal{W}$ à partir du graphe G^**

[2]Dans le but de simplifier la transformation polynomiale, nous avons privilégié l'approche créer des tâches et des contraintes de précédence quite à les supprimer ensuite que de lister de manière fastidieuse les contraintes de précédence.

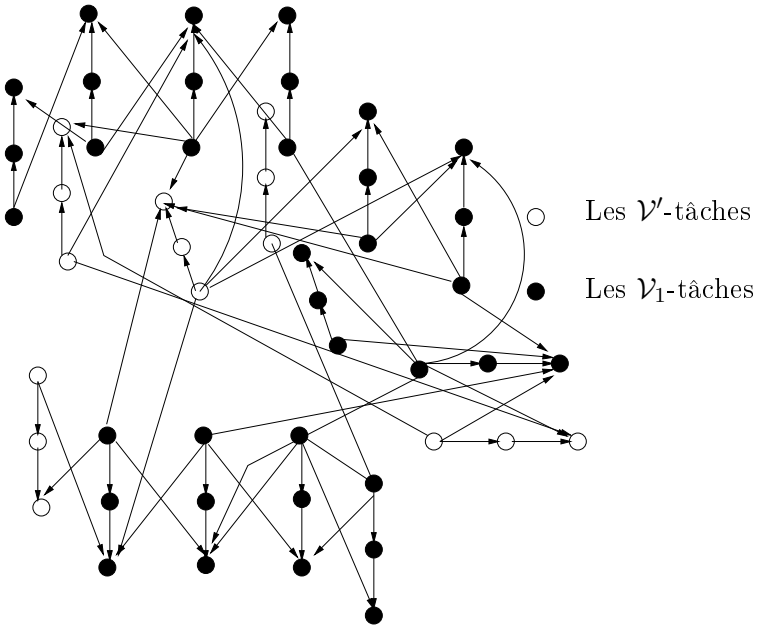


Fig. 6.13 – *Suite de la construction de $\mathcal{W}$. Création des chaînes de longueur trois et ajout des contraintes de précédence entre les tâches*

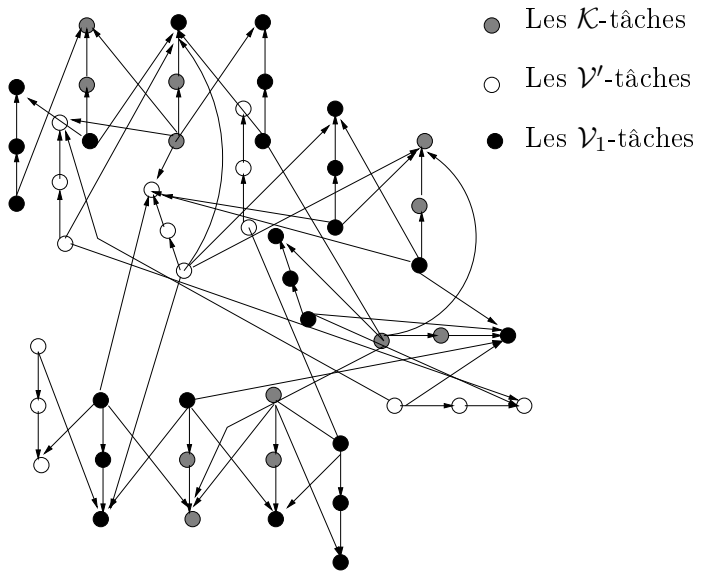


FIG. 6.14 – *Partition du graphe G^* en fonction des ensembles de sommets $\mathcal{V}_1$, $\mathcal{K}$ et $\mathcal{V}$*

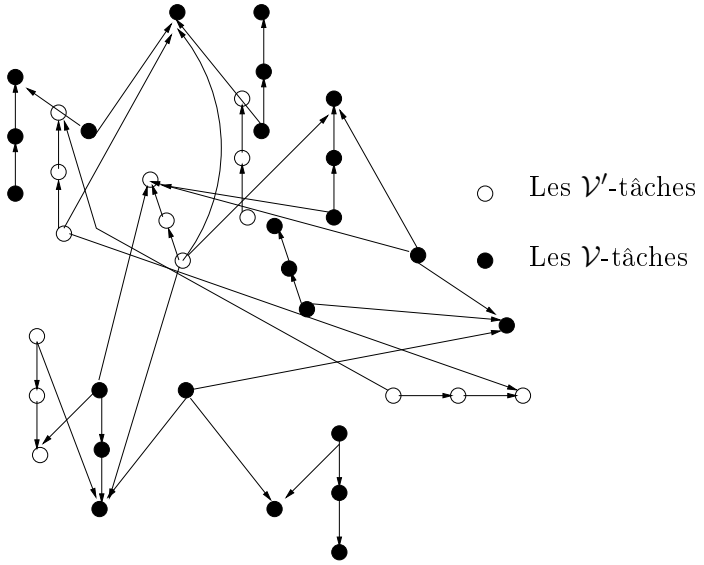


FIG. 6.15 – *Graphe $\mathcal{W}$ résultant issu des diverses transformations*

Les figures 6.12, 6.13, 6.14, 6.15 illustrent la construction du graphe $\mathcal{W}$ à partir d'un graphe $G^* \in \mathcal{G}$.

Pour compléter la transformation, le nombre de processeurs est $m = n^*$, et ils sont numérotés π^i avec $i \in [1, n^*]$.

En résumé, le graphe de précédence $\tilde{G}$ est composé du graphe $\mathcal{W} = (\mathcal{V} \cup \mathcal{V}', E_{\mathcal{W}})$ avec $3n^* - 4nB$ tâches et les contraintes de précédence proposées ci-dessus et du graphe $\mathcal{Z} = \{\bigcup_{a_j \in \mathcal{A}} Z^{s(a_i)}\}$ avec $4nB$ tâches.

L'instance de notre problème et l'instance de n-PARTITION sont de tailles polynomiales (voir [67]). La réduction précédente est clairement une transformation polynomiale dépendant de B.

- Supposons $\mathcal{A} = \{a_1, \ldots, a_{3n}\}$ forment une partition en n sous-ensembles $\mathcal{A}_1, \ldots, \mathcal{A}_n$ ayant chacune une taille de B. Nous allons prouver qu'il existe un ordonnancement de longueur trois.
  Premièrement, la tâche $v_i^*[j] \in \mathcal{V}' \cup \mathcal{V}$ est exécutée sur le processeur π^i à $t = j$ avec $j \in \{0, 1, 2\}$ (si cette tâche existe).
  Considérons les processeurs sur lesquels les tâches de l'ensemble $\mathcal{V}$ sont ordonnancées. Par l'affectation précédente, ces processeurs sont numérotés par $\pi^1, \ldots, \pi^{2n(B+1)}$.
  Soit $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ une partition de $\mathcal{A}$. Nous considérons $\mathcal{A}_i = \{a_{i_1}, a_{i_2}, a_{i_3}\}$ avec i fixé. Les tâches du graphe $Z^{s(a_j)}, a_j \in \mathcal{A}_i$ sont exécutées entre les processeurs $\pi^{1+2(i-1)(B+1)}$ et $\pi^{2i(B+1)}$. De plus, les tâches $Z^{s(a_j)}[l, k], k \in \{0, 1\}, l \in J_0$ (resp. $k \in \{1, 2\}, l \in J_1$) sont ordonnancées sur $2s(a_{i_j})$ processeurs consécutifs dans le but de respecter un ordonnancement de longueur trois.
  Ainsi sans perte de généralité, nous pouvons supposer que les tâches du graphe $Z^{s(a_{i_1})}$ sont ordonnancées entre les processeurs $\pi^{1+2(i-1)(B+1)}$ et $\pi^{2(i-1)(B+1)+2s(a_{i_1})}$. De manière similaire, les tâches de $Z^{s(a_{i_2})}$ (resp. $Z^{s(a_{i_3})}$) sont exécutées entre les processeurs $\pi^{2+2(i-1)(B+1)+2s(a_{i_1})}$ et $\pi^{1+2(i-1)(B+1)+2s(a_{i_1})+2s(a_{i_2})}$ (resp. $\pi^{2+2(i-1)(B+1)+2s(a_{i_1})+2s(a_{i_2})}$ et $\pi^{2i(B+1)}$).
- Supposons qu'il existe un ordonnancement S de longueur trois. Nous allons prouver que $\mathcal{A} = \{a_1, \ldots, a_{3n}\}$ peut être partionné en n sous-ensembles disjoints $\mathcal{A}_1, \ldots, \mathcal{A}_n$ de somme chacune égale à B.

Lemme 6.5.2 *Dans n'importe quel ordonnancement valide de longueur trois, il n'existe aucun temps d'inactivité.*

Preuve
Le nombre de processeurs est $m = n^*$ et le nombre de tâches à ordon-

nancer est $3n^*$ ($4nB$ pour le graphe $\mathcal{Z}$ et $3n^* - 4nB$ pour le graphe $\mathcal{W}$). Donc tous les processeurs sont actifs. □

Lemme 6.5.3 *Dans n'importe quel ordonnancement valide de longueur trois, les tâches issues du sous-graphe engendré par les $\mathcal{V}$-tâches doivent être exécutées sur $2(B+1)$ processeurs consécutifs.*

Preuve
Considérons le sous-graphe engendré par les $\mathcal{V}$-tâches. Ce graphe de précédence admet des chemins de longueurs deux et unitaire (en nombre d'arcs). Il est clair que les chemins de longueur deux ne peuvent être exécutés que sur le même processeur.
Considérons les tâches associées aux chemins de longueur un. Soit $v_i^*[0] \in \mathcal{V}$ une tâche sans prédécesseur. Par construction $v_i^*[0]$ admet un successeur noté $v_{i+1}^*[2] \in \mathcal{V}$.
Supposons que ces deux tâches sont affectées sur le même processeur π^l. Sachant que $v_{i+1}^*[2]$ admet un autre prédécesseur, noté $v_{i+2}^*[0] \in \mathcal{V}$ alors $v_{i+1}^*[2]$ est nécessairement exécutée à $t = 2$.
La tâche $v_{i+2}^*[0]$ ne peut être exécutée à $t = 1$ sur π^l sachant que cette tâche admet un autre successeur que $v_{i+1}^*[2]$. Ainsi il existe un slot de libre à $t = 1$ sur le processeur π^l. Sachant qu'il n'existe aucune tâche indépendante, que le graphe $\mathcal{Z}$ admet que des chemins de longueur un, alors ce slot restera libre. Impossible.
En conclusion le sous-graphe induit par les $\mathcal{V}$-tâches doit être exécuté sur $2(B+1)$ processeurs consécutifs.

□

Lemme 6.5.4 *Dans n'importe quel ordonnancement valide de longueur trois, les deux sous-graphes engendrées les $\mathcal{V}$-tâches appartenant à deux chemins disjoints de longueur $2(B+1)$ ne peuvent s'exécuter sur les mêmes processeurs.*

Preuve
En utilisant des arguments similaires à ceux utilisés pour la démonstration du Lemme 6.5.3, nous pouvons aisément démontrer le lemme.
□

Lemme 6.5.5 *Dans n'importe quel ordonnancement valide de longueur trois $Z^{s(a_j)}$-tâches doivent être exécutées sur les mêmes processeurs que les $\mathcal{V}$-tâches.*

Preuve
Soit $\Pi = \{\pi^l | \mathcal{V} - \text{tâches qui sont ordonnancées sur } \pi^l\}$ un ensemble de processeurs pour lesquels les $\mathcal{V}$-tâches sont exécutées. Supposons que l'ensemble des $Z^{s(a_j)}$-tâches sont exécutées sur les processeurs avec $\pi^k \notin \Pi$. Par le lemme 6.5.2, il n'existe pas de temps d'inactivité alors les tâches des chemins de longueur trois sont nécessairement exécutées sur un processeur $\pi^* \in \Pi$. Ceci est impossible par le lemme 6.5.3.

□

Par les lemmes précédents, nous savons que les $6n(B+1)$ tâches (les $\mathcal{V}$-tâches et les $Z^{s(a_j)}$-tâches) sont ordonnancées sur les n chemins disjoints de longueur $2B+1$. Par la Définition 6.5.2, nous savons que le graphe G^* admet un unique ensemble de n chemins disjoints de longueur $2B+1$) avec les propriétés requise. De plus, du fait des précédence entre ces tâches il est clair que ces tâches sont exécutées sur un chemin de processeurs de longueur $(2B+1)$.
Construisons une partition $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ avec les propriétés désirées à partir de l'ordonnancement S de longueur trois. Nous savons que deux tâches du même sous-graphe $Z^{s(a_j)}$ (voir le Lemme 6.5.5) ne peut être exécutées sur deux chemins différents. La distance en nombre d'arêtes entre ces deux processeurs est au moins de deux.
Nous définissons $\mathcal{A}_\ell$ tel que chaque élément $a_j \in \mathcal{A}_\ell$ si et seulement si les tâches de graphe $Z^{s(a_j)}$ sont ordonnancées entre les processeurs numérotés $\pi^{1+(j-1)2(B+1)}$ à $\pi^{2j(B+1)}$ avec un j fixé.
Maintenant, nous allons calculer $\sum_{a_i \in \mathcal{A}_\ell} s(a_i)$. En se basant sur les remarques précedentes, et sans perte de généralité, nous pouvons supposer que $v_i^*[k]$ avec $i\{1, \ldots, 2n(B+1)\}$ et $k \in \{0,1,2\}$ (si elle existe) peut-être affecté au π^l. Considérons les $Z^{s(a_j)}$-tâches qui sont exécutées entre les processeurs $\pi^{1+(j-1)2(B+1)}$ et $\pi^{2j(B+1)}$ pour un j fixé.
En utilisant le Lemme 6.5.3, nous savons que le nombre de $\mathcal{V}$-tâches qui sont ordonnancées sur les processeurs $\pi^{1+(j-1)2(B+1)}$ et $\pi^{2j(B+1)}$ pour un j fixé est de $6+2B$.
Pour finir, nous pouvons conclure que $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ forme une de $\mathcal{A}$ avec les propriétés requises.

□

Nous pouvons étendre le résultat précédent, comme pour la topologie en étoile aux grands délais de communication i.e. $c_{ij} = \theta$.

Corollaire 6.5.1 *Il n'existe pas d'algorithme approché avec une perfor-*

mance relative inférieure à $1+\frac{1}{\theta+2}$ *(à moins que* $\mathcal{P}=\mathcal{NP}$*) pour le problème* $(P,G^*)|prec, c_{ij}=\theta; d(\pi^*,\pi^j)=1; p_i=1|C_{max}$.

Corollaire 6.5.2 *Il n'existe pas d'algorithme approché avec une performance relative inférieure à* $1+\frac{1}{\theta+2}$ *(à moins que* $\mathcal{P}=\mathcal{NP}$*) pour le problème* $(P,G^*)|prec, c_{ij}=\theta; d(\pi^k,\pi^j)=1, p_i=1, dup|C_{max}$.

6.5.3 Discussion

Dans la section précédente, nous avons proposé une classe de graphes pour laquelle le problème de décider si une instance de $(P,G^*)|\beta; c_{ij} = d(\pi^\ell,\pi^k) \leq d;\ p_i=1|C_{max}$ admet un ordonnancement de longueur au plus trois est $\mathcal{NP}$-complet avec $\beta \in \{prec, bipartite\}$ et $G^* \in \mathcal{G}$. Utilisons ce résultat pour montrer que pour les topologies suivantes le problème est $\mathcal{NP}$-complet.

- Pour une grille (où le tore), il suffit d'avoir $2n+1$ lignes et $2B+2$ colonnes,
- Pour un arbre binaire complet, il suffit de considérer un arbre de hauteur $\lceil \log(n) \rceil + 2B + 1$,
- Pour l'hypercube $H(d)$ (où le Cube Connected Cycles), il est suffisant de choisir $d = 2\lceil \log(n) \rceil + B + 2$.
- ...

Remarquons que dans le cas de l'étoile (le diamètre est constant), le problème devient $\mathcal{NP}$-complet pour $C_{max}=5$ tandis que pour les topologies dont le diamètre est un paramètre de l'instance le problème devient $\mathcal{NP}$-complet pour $C_{max}=3$.

6.5.4 Algorithme d'approximation pour un graphe structuré diamètre d

Nous proposons maintenant un algorithme d'approximation pour un graphe structuré de diamètre d. Nous utilisons la diamètre d comme borne supérieure sur les communications et nous considérons le problème avec des grands délais de communication.

Théorème 6.5.2 *Le problème* $(P,G^*)|prec;\ c_{ij}=d; d(\pi^k,\pi^l); p_i=1|C_{max}$ *admet un algorithme avec une performance garantie de* $\frac{(d+1)^2}{3}+1$.

Preuve

G^*	G		Complexité		Borne d'approx.	
Topologie	Précédence	c_{ij}	Poly.	$\mathcal{NP}$-complet	Inf.	Sup.
Chaîne	$prec, biparti$	1	2	3	4/3	$1+\frac{2m}{3}$
	$prec, biparti$	d	$d+1$	$d+2$	$\frac{d+3}{d+2}$	$\frac{4}{3}\frac{(d+1)^2}{(d+2)}+1$
Cycle	$prec, biparti$		1 2	3	4/3	$1+\frac{2m}{3}$
	$prec, biparti$	d	$d+1$	$d+2$	$\frac{d+3}{d+2}$	$\frac{4}{3}\frac{(d+1)^2}{(d+2)}+1$
Étoile	$prec, biparti$	1	4	5	6/5	4
	$prec, biparti$	d	$2d+2$	$2d+3$	$\frac{2d+4}{2d+3}$	$\frac{2}{3}(\frac{(d+1)^2}{d+2}+1$
$G^* \in \mathcal{G}$	$prec, biparti$	1	2	3	4/3	$\frac{25}{9}$
	$prec, biparti$	d	$d+1$	$d+2$	$\frac{d+3}{d+2}$	$\frac{4}{3}\frac{(d+1)^2}{(d+2)}+1$

TAB. 6.2 – Résumé des nouveaux résultats montrés sur ce modèle. Notons que $G^* \in \{Arbre\ binaire, Grille, Tore, Connected\ Cube\ cycle, \ldots\}$. La distance $d(\pi^l, \pi^h)$ est le nombre d'arcs dans le graphe des processeurs G^*

Il suffit d'étendre les résultats obtenu pour l'étoile avec un coefficient de dilatation $(d+1)/2$ et d'utiliser le lemme 6.4.1 avec la valeur de $(d+1)/2$). Le facteur $\frac{4}{3}$ vient toujours de l'algorithme de Munier et König [115]. La valeur additionnelle est issue du lemme 6.2.1.

□

6.6 Conclusion

Nous avons étudié un modèle qui prend en compte le placement des tâches sur les processeurs comme une caractéristique fondamentale. Nous avons étudié la complexité sur divers topologies et nous avons développé plusieurs algorithmes approchés avec garantie de performance. Les ratios trouvés dans le pire des cas dépendent du nombre de processeurs comme dans le cas du `modèle homogène` pour les problèmes avec grands délais de communication. En effet, les problèmes étudiés peuvent être considérés comme des problèmes du `modèle homogène` avec des délais de communication égaux à m, le nombre de processeurs.

Dans le cas spécifique de l'étoile, nous avons procédé à une étude complète de la complexité (nous n'avons pas de « trou », pour toutes valeurs du C_{max} nous avons classifié le problème) et développer un algorithme d'approximation avec un ratio constant.

En dernier lieu, nous proposons une classe de graphes $\mathcal{G}$ pour laquelle

le problème de décider si nous pouvons ordonnancer un graphe appartenant à cette classe en temps temps trois au maximum est $\mathcal{NP}-$complet. Cette classe est assez grande pour contenir les arbres, les grilles, tores, ...

Il serait intéressant de poursuivre la recherche d'algorithmes d'approximation pour les topologies proposées :

- Soit en se basant sur les algorithmes approchés qui ont été développés dans le cas du `modèle homogène`.
- Soit en développant de nouveaux algorithmes spécifiques en prenant en compte la topologie.

En dernier lieu, dans les hypothèses décrites dans l'introduction, nous avons retenu comme fonction de distance celle qui donne la longueur d'un plus court chemin. Nous pouvons considérer une autre fonction : une fonction de routage dans le graphe de processeurs. Ce choix est nécessaire lorsque nous devons prendre en compte le phénomène de congestion sur les liens. En effet, un lien de communication peut supporter qu'une certaine charge c'est-à-dire une quantité limitée de paquets peut transiter sur ce lien à un instant donné. Ces contraintes de charges obligent à dérouter certains paquets par rapport à la route initialement prévue. Ainsi, la fonction de distance basée sur l'algorithme des plus courts chemins n'est plus adaptée. Il est donc nécessaire de proposer un algorithme qui prend en compte les caractéristiques du réseau.

Troisième partie

Applications des problèmes d'ordonnancement

CHAPITRE

7 Communications exactes et incompressibles

7.1 Introduction

Nous sommes également intéressés à deux problèmes d'ordonnancement ayant potentiellement des applications pratiques. Le premier concerne le problème d'ordonnancer des données d'acquisition pour une torpille en immersion tandis que le second se focalise sur le problème d'ordonnancer des tâches viticoles dans un domaine incertain (lié aux manques de données météorologiques sur le long terme ; par exemple la semaine) .

Le chapitre se décompose de la manière suivante :

- dans la section 7.2, nous procéderons à une analyse de la complexité et de l'approximation pour le problème d'acquisition de données pour une torpille en immersion. Ces travaux ont été mené avec J.C. König, B. Darties et G. Simonin (voir [153], [155], [156], [157], [158], [159], [160],[161], [154], [162]).
- dans la section 7.3, nous procéderons à la présentation assez succincte de la problématique liée à l'ordonnancement de tâches viticoles. En effet, nous avons abordé le problème avec une approche tournées de véhicules. Ces travaux ont été menés avec D. Feillet, F. Hernandez, J.C. König, B. Léger, O. Naud, T. Tuitete (voir [83], [84], [85], [117], [118], [119].).

7.2 Le problème d'acquisition de données pour une torpille en immersion

7.2.1 Motivations

Ce travail s'inscrit dans le cadre d'une collaboration avec David Andreu du Département Robotique du LIRMM. Nous nous intéressons à des solutions algorithmiques pour l'acquisition de données pour une torpille en immersion. 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 capables de se déplacer seuls et de mener à bien des tâches qui nécessitent encore l'assistance de l'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 état et l'état de son environnement, les combiner avec la mission qui lui a été confiée et prendre des décisions cohérentes.

Les roboticiens du LIRMM travaillent sur une torpille appelé Taipan c'est un véhicule sous-marin totalement autonome. Autonomie de déplacements, de navigation, la torpille calcule en permanence ses déplacements à l'aide des informations délivrées par des capteurs de pression, d'accélération, de vitesse et d'inclinaison lui permettant de connaître sa profondeur d'immersion. 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 marins. Les données collectées sont mémorisées dans Taipan.

Nous avons décidé d'aborder ce problème en utilisant la notion de tâches-couplées, introduite en premier par Shapiro [150] pour l'acquisition de données radar, soumises à des contraintes de compatibilité en présence ou non de tâches de traitement. La justification de l'utilisation des tâches-couplées est donnée dans [152]. Le but de ce travail fut d'évaluer l'impact de l'introduction du graphe de compatibilité sur la complexité et l'approximation dans les problèmes d'ordonnancement de tâches-couplées.

L'organisation de cette partie est la suivante :

- La section 7.2.2 sera dédiée aux définitions et aux hypothèses,
- La section 7.2.4 sera consacré aux résultats de complexité et d'approximation pour le problème des tâches-couplées en présence d'un graphe de compatibilité complet,
- Les résultats de complexité seront présentés sous-forme de treillis dans la partie 7.2.5.

– Des algorithmes d'approximation seront présentés dans la partie 7.2.6.

7.2.2 Description des tâches d'acquisition et de traitement

Nous caractérisons notre problème en utilisant deux types de tâches ayant des caractéristiques très fortes.

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

Définition 7.2.1 (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* $t_{A_i} = t_{a_i}$ *et la date de complétion* $C(A_i) = t_{a_i} + a_i + L_i + b_i$ *(voir Figure 7.1).*

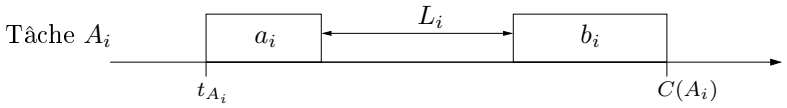


Fig. 7.1 – *Illustration d'une tâche d'acquisition* A_i

Par la suite, nous décrirons les tâches 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 et la variable L_i représente le temps d'inactivité entre l'exécution de ces deux sous-tâches.

Définition 7.2.2 (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. 7.2 – *Illustration d'une tâche de traitement $\mathcal{T}_i$*

Nous supposerons dans la suite que chaque tâche d'acquisition admet une et une seule tâche de traitement comme successeur et de manière symétrique une tâche de traitement possède une seule tâche d'acquisition comme prédécesseur, sauf mention contraire.

Les tâches d'acquisition vont avoir un autre type de contraintes comme nous l'avons évoqué 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 7.2.3 (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 7.3).*

La présence d'un graphe de compatibilité sera noté par G_c. Il est important de noter que lorsque G_c est un graphe complet nous retrouvons les problèmes classiques de tâches-couplées.

Définition 7.2.4 (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 7.3). Les tâches A_i et A_j peuvent alors se chevaucher lors de leur exécution.*

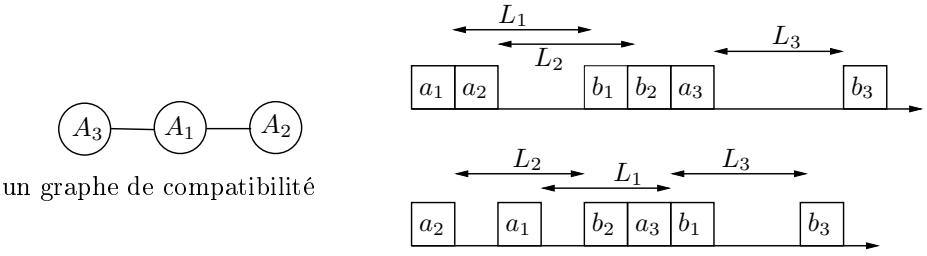


FIG. 7.3 – *Illustration de la contrainte d'incompatibilité*

p_{a_i}	$p_{a'_i}$	L_i	Complexité	Référence
$a'_i = L_i$	$a_i = L_i$	$a_i = a'_i$	$\mathcal{NP}$-complet	[122]
a_i	L	b	$\mathcal{NP}$-complet	[122]
a	L	b_i	$\mathcal{NP}$-complet	[122]
p	L_i	p	$\mathcal{NP}$-complet	[122]
1	L	1	$\mathcal{NP}$-complet[2]	[17]
a	L	b	Pseudo-polynomial	[6]
p	p	b_i	Polynomial	[122]
a_i	p	p	Polynomial	[122]

TAB. 7.1 – Résumé des résultats de complexité pour les tâches-couplées avec un graphe de compatibilité complet

7.2.3 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 :

- $t_{b_i} = t_{a_i} + a_i + L_i, \forall A_i \in \mathcal{A}$ où t_{a_i} désigne la date de début d'exécution de la sous-tâche a_i.[1]
- $t_{\tau_i} \geq C(b_j)$ (resp. $t_{\tau_j} \geq C(\tau_i, \forall A_j, \mathcal{T}_i$ (resp. $\forall \mathcal{T}_i, \mathcal{T}_j$) 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.
- $t_{a_i} \notin [t_{a_j}, C(b_j)[$ et $t_{b_i} \notin [t_{a_j}, C(b_j)[$, $\forall i$, $\forall j$ tels que A_i et A_j sont incompatibles.

Au lieu d'utiliser la notation à trois champs $\alpha|\beta|\gamma$ classique, nous noterons le problème de tâches-couplées $(a_i, L_i, b_i) \cup \tau_i$.

7.2.4 Complexité et approximation pour les tâches couplées en présence d'un graphe de compatibilité complet

La plupart des résultats de complexité ont été obtenus par Oman et Potts [122].

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

[2] Il existe des contrinets de précédence entre les tâches. Les contraintes de précédence entre les tâches sont strictes c'est-à-dire si $A_i \rightarrow A_j$ alors $a'_i \rightarrow a_j$

p_{a_i}	$p_{a'_i}$	L_i	Approximation	Seuil	Référence
$a_i \leq a'_i$	L_i	a'_i	7/2	$2-\epsilon$	[4]
a_i	L_i	$a'_i \leq a_i$	3	$2-\epsilon$	[4]
$a_i = a'_i$	L_i	$a_i = a'_i$	5/2	$2-\epsilon$	[4]
$a_i = a'_i = 1$	L_i	$a_i = a'_i = 1$	1.75		[3]

TAB. 7.2 – Résumé des résultats d'approximation pour les tâches-couplées

Les tableaux 7.1 et 7.2 récapitulent les résultats de complexité et d'approximation sur les tâches-couplées. Il est important de noter que dans tous les résultats (en complexité et en approximation) le graphe de compatibilité est absent (ou ce qui est équivalent de dire que le graphe de compatibilité est complet).

7.2.5 Résultats de complexité sur les tâches-couplées en présence d'un graphe de compatibilité

Plusieurs résultats de complexité ont été obtenus selon les caractéristiques du problème de tâches-couplées sous-jacent. Les résultats sont classifiés sous forme d'un treillis d'inclusion des problèmes des « plus facile », et sont donnés par les figures 7.4, 7.5, 7.6. Une arête dans un treillis représente une symétrie entre les problèmes et un arc une inclusion.

Théorème 7.2.1 ***Tiré de [152]***

En se basant sur les treillis donné par la figure 7.4, nous pouvons montrer que les problèmes donnés ci-après sont $\mathcal{NP}$-complets :

- Π_1 *:* $1|a_i = b_i = p, L_i = L, G_c|C_{max}$,
- Π_3 *:* $1|a_i = a, L_i = L, b_i = b, G_c|C_{max}$,
- Π_2 *:* $1|a_i = L_i = b_i, G_c|C_{max}$.

Preuve

La preuve de la difficulté, au sens de la complexité pour le problème Π_1 : $1|a_i = b_i = p, L_i = L, G_c|C_{max}$, se fait à partir du problème Partition en Triangles (voir Section 2.5).

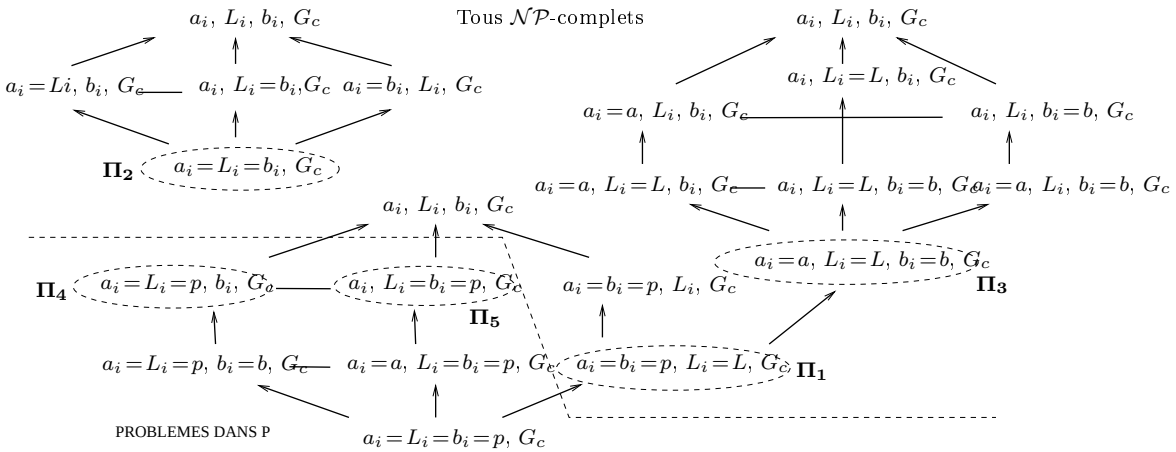


FIG. 7.4 – *Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition en présence de contraintes d'incompatibilité*

Celle de Π_3 est évidente par inclusion des paramètres du problème d'ordonnancement de Π_1 dans Π_3.

Pour finir le problème Π_2 est $\mathcal{NP}$-complet sachant que la version avec un graphe de compatibilité complet est déjà $\mathcal{NP}$-complet (voir [122]).

□

Théorème 7.2.2 ***Tiré de [152]***

Les problèmes suivants sont polynomiaux.

- Π_4 : $1|a_i = L_i = p, b_i, G_c|C_{max}$
- Π_5 : $1|a_i, L_i = b_i = p, G_c|C_{max}$

Preuve

Les preuves de la polynomialité des problèmes précédents se trouvent dans [152]. Il est intéressant de noter que les preuves sont astucieuses et qu'elles utilisent la notion de couplage de taille maximum dans un graphe non orienté.

□

Théorème 7.2.3 ***Tiré de [152]***

En se basant sur les treillis donné par la figure 7.4, nous pouvons montrer que les problèmes donnés ci-après sont $\mathcal{NP}$-complets :

- Π_1^P : $1|prec, (a_i = b_i = p, L_i) \cup (\tau_i \geq 1, pmtn), G_c$ *complet*$|C_{max}$,
- Π_2^P : $1|prec, (a_i = b_i = L_i) \cup (\tau_i \geq 1, pmtn), G_c$ *complet*$|C_{max}$,

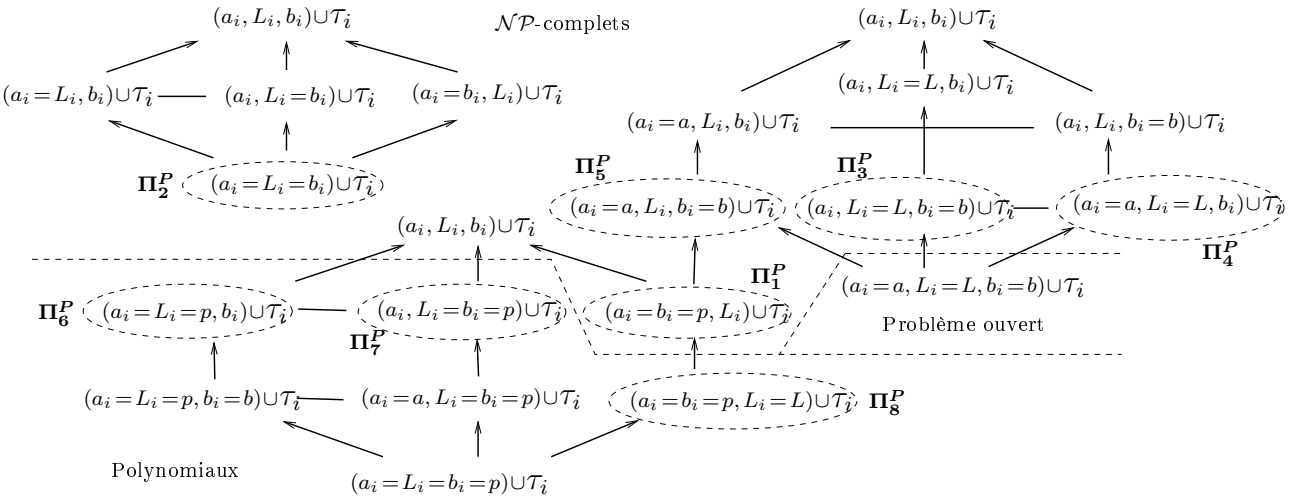


FIG. 7.5 – *Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition en présence de contraintes de précédence*

- Π_3^P : $1|prec, (a_i, L_i = L, b_i = b) \cup (\tau_i \geq 1, pmtn), G_c$ *complet*$|C_{max}$,
- Π_4^P : $1|prec, (a_i = a, L_i = L, b_i) \cup (\tau_i \geq 1, pmtn), G_c$ *complet*$|C_{max}$,
- Π_4^P : $1|prec, (a_i = a, L_i, b_i = b) \cup (\tau_i \geq 1, pmtn), G_c$ *complet*$|C_{max}$,

Preuve

Les preuves sont similaires à celles données dans [122].

□

Théorème 7.2.4 ***Tiré de [152]***

Les problèmes suivants sont polynomiaux.

- Π_6^P : $1|prec, (a_i = L_i = p, b_i = b) \cup (\tau_i, pmtn), G_c$ *complet*$|C_{max}$
- Π_7^P : $1|prec, (a_i = a, L_i = b_i = p) \cup (\tau_i, pmtn), G_c$ *complet*$|C_{max}$
- Π_8^P : $1|prec, (a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c$ *complet*$|C_{max}$

Preuve

Les preuves de la polynomialité des problèmes précédents se trouvent dans [152]. Il est intéressant de noter que les preuves sont techniques du moins pour le problème Π_6^P, et utilisent les arguments fondamentaux et classiques sur monoprocesseur. Pour valider une propriété nous procédons à des échanges sur l'emplacement des tâches et nous raisonnons par l'absurde.

□

Théorème 7.2.5 ***Tiré de [152]***

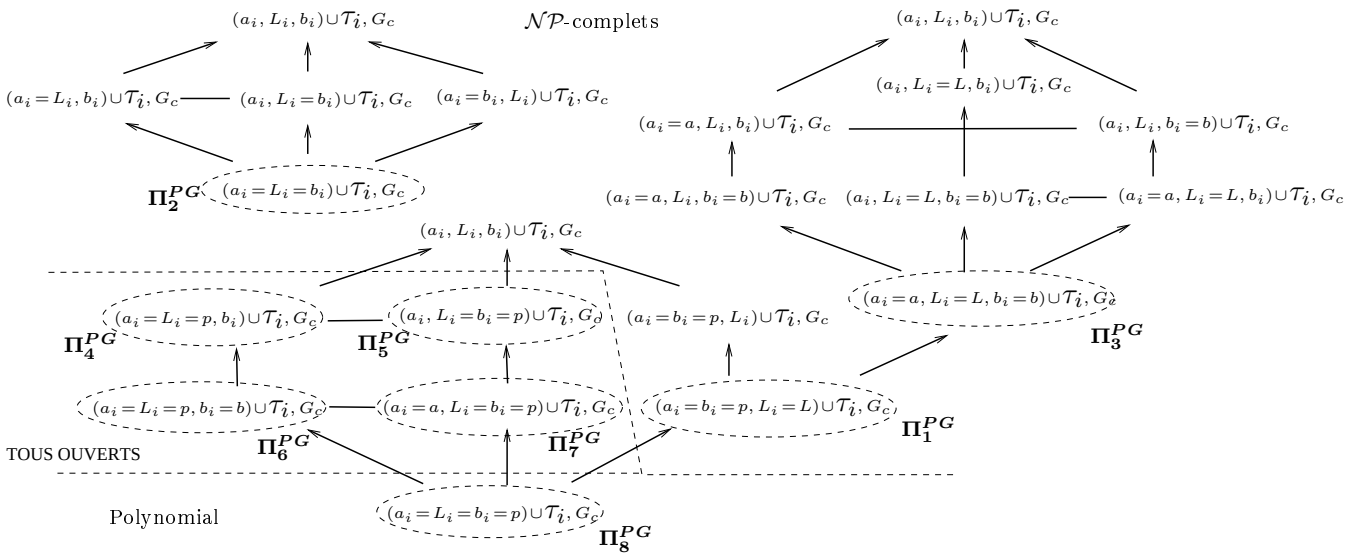


FIG. 7.6 – *Visualisation globale de la complexité des problèmes d'ordonnancement avec tâches d'acquisition et tâches de traitement en présence de contraintes de précédence et d'incompatibilité*

En se basant sur les treillis donné par la figure 7.4, nous pouvons montrer que les problèmes donnés ci-après sont $\mathcal{NP}$-complets :

- Π_1^{PG} *:* $1|prec, (a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c$ *complet*$|C_{max}$,
- Π_3^{PG} *:* $1|prec, (a_i = a, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c$ *complet*$|C_{max}$,

Preuve

La preuve de Π_1^{PG} se fait à partir du problème $\mathcal{NP}$-complet clique de taille maximum. Celle de Π_3^{PG} est évidente par inclusion des paramètres du problème d'ordonnancement de Π_1^{PG} dans Π_3^{PG}.

□

Remarque 7.2.1 ***Tiré de [152]***

Les problèmes suivants sont ouverts.

- Π_4^{PG} *:* $1|prec, (a_i = L_i = p, b_i) \cup (\tau_i, pmtn), G_c|C_{max}$
- Π_5^{PG} *:* $1|prec, (a_i, L_i = b_i = p) \cup (\tau_i, pmtn), G_c|C_{max}$
- Π_6^{PG} *:* $1|prec, (a_i = L_i = p, b_i = b) \cup (\tau_i, pmtn), G_c|C_{max}$

Les problèmes précédents n'ont pas été encore classifiés au sens de la complexité, ceci tiens du fait que nous ne sommes pas en mesure pour le moment de caractériser une solution optimale. Intuitivement, nous pouvons

penser pour Π_4^{PG}, que la recherche d'un couplage de taille maximum produit un ordonnancement optimal. Ceci n'est pas le cas, il suffit de considérer l'instance donné par la figure 7.7. De manière similaire, pour Π_6^{PG}, nous exhibons un contre-exemple dans lequel, l'exécution par ordre décroissant des poids des arêtes, ne conduit pas à un ordonnancement optimal (voir la figure 7.8).

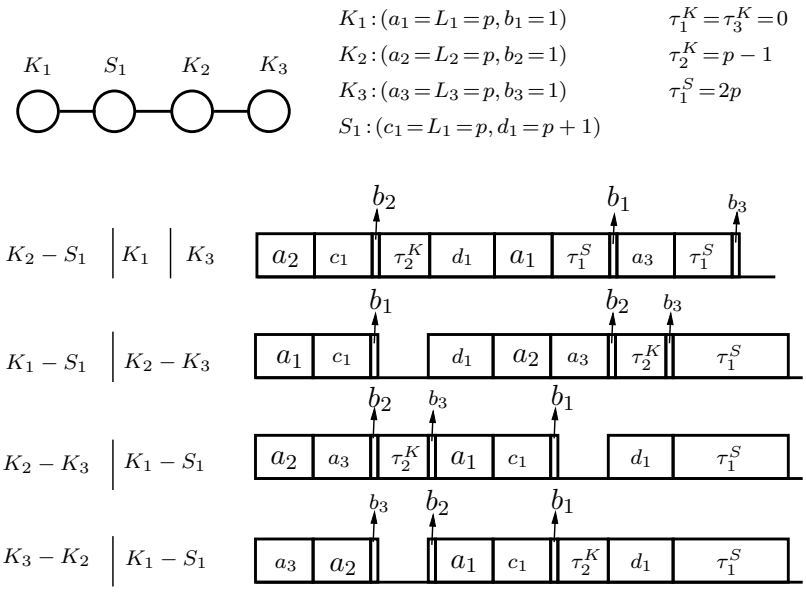


FIG. 7.7 – *Un contre-exemple pour le problème Π_4^{PG}. Il est intéressant de noter que dans le cas où le graphe de compatibilité est complet le problème devient polynomial en utlisant la recherche d'un couplage de taille maximum.*

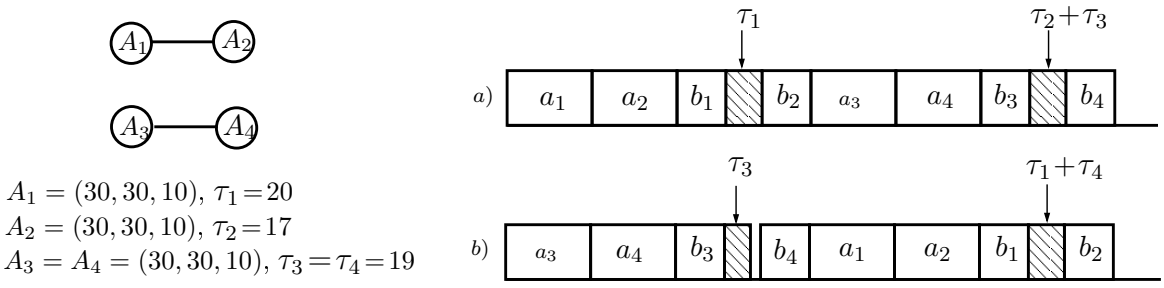


FIG. 7.8 – *Un contre-exemple pour le problème Π_6^{PG}. Chaque tâche A_i est équivalente au modèle $(a_i = L_i = p, b_i = b)$. Pour un couplage donné, l'ordonnancement b) exécute les tâches de l'arête de poids maximum en premier mais crée un slot, alors que dans le cas a) l'ordonnancement est sans temps d'inactivité si nous n'exécutons pas la tâche de poids maximum en premier.*

7.2.6 Résultats d'approximation sur les tâches-couplées en présence d'un graphe de compatibilité

Cette partie est dévolue à l'étude de plusieurs algorithmes d'approximation avec un ratio non trivial.

Le tableau 7.3 synthétisent les résultats obtenus sur les tâches-couplées en présence d'un graphe de comptabilité. La plupart des résultats d'approximation sont basés sur le recherche d'un couplage de taille maximum dans le graphe de compatibilité ou d'une clique de taille maximale. Nous avons dû également étendre la notion de couplage de taille maximum en couvrant le graphe de compatibilité par des chaînes de longueur au plus deux (en nombre d'arcs) dans le but de minimiser le nombre de sommets non couverts. Nous notons ce problème le 2-recouvrement. Nous avons étendu la notion de chaîne améliorante (notion fondamentale en théorie des graphes) en prenant en compte les chaînes de longueur au plus deux. De plus, nous caractérisons la notion de 2-recouvrement de taille maximum. Nous proposons un algorithme de complexité quadratique pour obtenir une couverture du graphe par un 2-recouvrement de taille maximum. Pour finir, nous proposons une relation de dualité pour un 2-recouvrement via le théorème suivant :

Théorème 7.2.6 ***Tiré de [152]*** *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 :*

Description du problème	Ratio	Références
$\Pi_1 : 1\|a_i = b_i = p, L_i = L, G_c\|C_{max}$	$\rho \leq \frac{7p+L}{4p}$	[152]
$\Pi_2 : 1\|a_i = L_i = b_i, G_c\|C_{max}$	$\rho \leq \frac{3}{2}$	[152]
$\Pi_3 : 1\|a_i = a, L_i = a+b, b_i = b, G_c\|C_{max}$	$\rho \leq \frac{1+\sqrt{3}}{2}$	[152]
$\Pi_4 : 1\|a_i = L_i = p, b_i, G_c\|C_{max}$	$\rho = 1$	[152]
$\Pi_5 : 1\|a_i, L_i = b_i = p, G_c\|C_{max}$	$\rho = 1$	[152]
$\Pi_1^P : 1\|prec, (a_i = b_i = 1, L_i) G_c\ complet\|C_{max}$	$\rho \leq \frac{7}{4}$	[3]
$\Pi_2^P : 1\|prec, (a_i, b_i, L_i), G_c\ complet\|C_{max}$	$\rho \leq \frac{7}{2}$	[2]
$\Pi_3^P : 1\|prec, (a_i < b_i, L_i), G_c\ complet\|C_{max}$	$\rho \leq 3$	[2]
$\Pi_4^P : 1\|prec, (a_i > b_i, L_i) G_c\ complet\|C_{max}$	$\rho \leq 3$	[2]
$\Pi_4^P : 1\|prec, (a_i = b_i, L_i), G_c\ complet\|C_{max}$	$\rho \leq \frac{5}{2}$	[2]
$\Pi_6^P : 1\|prec, (a_i = L_i = p, b_i = b) \cup (\tau_i, pmtn), G_c\ complet\|C_{max}$	$\rho = 1$	[152]
$\Pi_7^P : 1\|prec, (a_i = a, L_i = b_i = p) \cup (\tau_i, pmtn), G_c\ complet\|C_{max}$	$\rho = 1$	[152]
$\Pi_8^P : 1\|prec, (a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c\ complet\|C_{max}$	$\rho = 1$	[152]
$\Pi_1^{PG} : 1\|prec, (a_i = b_i = p, L_i = L) \cup (\tau_i, pmtn), G_c\|C_{max}$	$\rho \leq \frac{L+6}{6}$	[152]
$\Pi_3^{PG} : 1\|prec, (a_i = a, L_i = L, b_i = b) \cup (\tau_i, pmtn), G_c\|C_{max}$	$\rho \leq \frac{3}{2}$	[152]
$\Pi_4^{PG} : 1\|prec, (a_i = L_i = p, b_i) \cup (\tau_i, pmtn), G_c\|C_{max}$	Ouvert	[152]
$\Pi_5^{PG} : 1\|prec, (a_i, L_i = b_i = p) \cup (\tau_i, pmtn), G_c\|C_{max}$	Ouvert	[152]
$\Pi_6^{PG} : 1\|prec, (a_i = L_i = p, b_i = b) \cup (\tau_i, pmtn), G_c\|C_{max}$	Ouvert	[152]

Tab. 7.3 – Tableau synthétisant les résultats d'approximation sur les tâches-couplées obtenus dans [152]

$$\max_S(|S| - 2|N(S)|) = \min_T K$$

7.2.7 Conclusion sur les tâches-couplées

Nous avons étudié du point de vue de la complexité et de l'approximation un problème issu d'une problématique concrète. Il est clair que pour un transfert des résultats vers les applications, nous devons prendre en compte d'autres paramètres qui ont été sciemment omis dans une première étude : les dates de disponibilité, date d'échéance, le caractère cyclique, développer des algorithmes efficaces et tester sur des données réelles ...

Nous pourrions également,

- Développer des algorithmes spécifiques (approchés à facteur constant ou optimaux selon leur classification) pour un graphe de compatibilité ayant une structure régulière (biparti, ...)
- Dans le problème originel, les tâches sont périodiques et admettent des dates de libération (date à partir de laquelle la tâche est ordonnançable) et une date d'échéance (date à laquelle la tâche doit être ordonnancée). Il est nécessaire de prendre en compte ces spécificités et de développer des solutions périodiques optimales.
- Certains résultats sur l'approximation montrent les ratios sont non constants et dépendent du paramètre L. La question fondamentale : « Est-ce que nous pouvons développer des algorithmes approchés avec des ratios α avec $\forall \alpha \in \mathbb{R}^{*,+}$ et indépendent des paramètres d'entrée ? » pour ces problèmes.

Une autre perspective serait de « plonger » ce problème dans un système temps réel : chaque tâche doivent être exécutées de manière répétitives et chaque exécution est contrainte par une borne inférieure (date de réalisation) et une borne supérieure (date d'échéance). Les dates de réalisation sont distribuées uniformément tout le long du temps avec égale distance appelée tâche période. La date d'échéance coïncide usuellement avec la date de réalisation de la période suivante. Et le but serait d'évaluer le temps total d'utilisation du processeur $W = \sum_{i=1}^{n} p_i/\pi_i$ où π_i désigne la période requise.

7.3 Le problème d'ordonnancement des tâches viticoles

Ces travaux ont été mené avec le chercheur Olivier Naud de l'équipe IODE de l'UMR ITAP du Cemagref de Montpellier.

De nos jours le respect de l'environnement est devenu une des priorités de nos états et a amené au concept de production agricole durable. Les exigences de cette production conduisent à des procédés de plus en plus complexes et la création de nouvelles règles de gestion devient nécessaire au bon fonctionnement de ces productions. Ceci est spécialement important pour le traitements des maladies de la vigne qui nécessite de nombreux traitements chimiques. Dans cette optique, des recherches sur les traitements phytosanitaires des vignes ont abouti à un ensemble de règles de décision permettant de réduire le nombre de traitements chimiques à effectuer par les exploitants. Mais l'application de ces règles peut avoir un impact significatif sur l'organisation du travail. La mise en place de ces règles doit donc satisfaire l'environnement mais également les viticulteurs. Nous devons mettre en place un processus de décision permettant de répondre à ces critères.

L'objectif du travail mené dans un premier temps (voir [167] et [81]) était d'élaborer une méthodologie d'évaluation de l'applicabilité temporelle de règles de décision, et des processus de mise en oeuvre associés. Après avoir vérifié la faisabilité via des méthodes de model-checking nous avons focalisé notre étude sur l'optimisation du processus.

Nous devons réduire le temps de travail au sein d'une exploitation composé de n parcelles, d'un ensemble F de m dépôts et de u opérateurs. Pour cela, nous devons déterminer l'ordre dans lequel effectuer un ensemble de traitements $\mathcal{T} = \{t_1, t_2, \ldots, t_h\}$, et grouper les parcelles i à traiter dans des lots de travail r_k. Dans le cas d'opérations de pulvérisation la taille maximale d'un lot dépend de la capacité de la cuve du pulvérisateur. Le temps total de travail varie en fonction de l'ordre dans lequel les parcelles sont traitées (uniquement du fait des temps de transport). Donc, la vérification de la faisabilité des traitements planifiés, suppose de proposer un ordonnancement couplé à un problème de tournées de véhicules.

Chaque traitement t_j est caractérisé par une consommation de ressource $d(t_j, i)$ et une durée d'exécution $st(t_j, i)$. Ces dernières diffèrent en fonction de la parcelle i traitée. Chaque parcelle doit être traitée dans un intervalle de temps, appelé par la suite fenêtre temporelle, qui est caractérisée par une borne inférieure (a_i, date au plus tôt) et une borne supérieure (b_i, date au plus tard). Le coût de déplacement c_{ij} entre deux parcelles est défini par la durée de déplacement entre ce deux parcelles. Nous supposerons que ces durées de déplacement respectent l'inégalité triangulaire.

A l'issue du calcul, chaque lot de travail r_k sera caractérisé par une durée et une quantité de ressources nécessaires à sa réalisation, correspondant respectivement à la somme des durées d'exécution et la somme des consommations de ressources des traitements effectués dans ce lot. Il sera également

caractérisé par une fenêtre temporelle $[a^k, b^k]$ durant laquelle il devra être effectué. Une fois tous les lots de travail effectués, toutes les parcelles devront avoir été traité une et une seule fois. La quantité de ressource nécessaire à chaque lot de travail ne devra pas dépasser une certaine limite Q correspondant à la capacité maximale de l'opérateur effectuant les traitements de ce lot. On peut rajouter une heuristique de capacité minimale de sorte que cette quantité de ressource ne soit en deçà de V_{coef} de cette limite. De plus, chaque lot de travail devra contenir au moins un dépôt $[f]$. Formellement, nous considérons le problème de tournée véhicule (en fait un tracteur) suivant :

Données :

- Un graphe $G = (V, E)$ avec $V = \{v_0, v_1, \ldots, v_n\}$ où v_0 désigne le dépôt, et $V \backslash \{v_0\}$ l'ensemble des clients (ici de parcelles),
- un coût c_{ij} pour tout arc $(v_i, v_j) \in E$,
- une demande d_i, une fenêtre temporelle $[a_i, b_i]$ et un temps de service st_i pour tout client $v_i \in V \backslash \{v_0\}$,
- une capacité de chargement,

Question : Trouver un ensemble de routes de coût minimum visitant tous les clients et respectant les contraintes de capacité et fenêtre temporelle, et tel que deux routes n'aient pas lieu en même temps.

Malgré ces similitudes, notre problème diffère du problème classique de routage de véhicules avec fenêtres de temps (VRPTW en anglais) et est en réalité un problème de routage de véhicules avec fenêtre de temps et routes multiple (Multi-trip Vehicle Routing Problem with Time Windows). Très peu de résultats proposant des solutions exactes, malgré l'intérêt pratique du problème ont été produits ([20] et [21]).

Cette particularité de pouvoir faire plusieurs routes avec un nombre de véhicules limités, augmente considérablement la complexité du problème et en change également sa modélisation.

Nous nous sommes intéressés à deux grandes classes de problèmes :

- Le problème avec limitation sur la durée des routes. Ce problème trouve un intérêt dans les applications où l'on dispose d'une flotte limitée de véhicules et, où, par exemple, les marchandises transportées sont périssables. Pour résoudre ce problème nous avons développé un algorithme à deux phases, dont la première porte sur la génération de toutes les structures (ou routes, ceci est possible car la limite de durée est basse) et la seconde est basée sur un schéma de Branch and Price avec une modification du processus classique de génération de colonnes. Cette méthode est très régulièrement employée dans les problèmes de tournées de véhicules avec fenêtre de temps. Elle permet de gérer implicitement

les solutions en ne générant effectivement qu'un ensemble restreint de solutions réalisables et potentiellement optimales. Elle résout alternativement un problème auxiliaire, générateur de solutions partielles, et un problème maître servant à déterminer une solution globale, voir [58] pour une description du principe de génération de colonnes. Dans ce cadre, nous avons procédé à plusieurs modifications du schéma classique de la génération de colonnes proposé dans le cadre des tournées de véhicules avec fenêtres de temps en incluant un processus d'ordonnancement [84], [83]. Ce processus d'ordonnancement a pour but de trouver un positionnement temporel des routes les unes par rapport aux autres. Dans ce cadre nous avons utilisé des stratégies classiques de complexité linéaire du type ASAP et ALAP.
Nous avons sensiblement les résultats obtenus par [20] et [21] grâce à une décomposition problème Maître/sous-problème différente et plus efficace.

- le problème sans limitation sur la durée des routes. Ce problème est une généralisation du problème précédent. Il est clair que la méthode précédente ne peut-être utilisée du fait de l'explosion combinatoire sur le nombre de routes générées pendant la phase 1. Dans ce cadre nous avons développé la première méthode exacte basée sur le schéma de Branch and Price avec pour routine la génération de colonnes pour ce problème. Afin de réduire la consommation mémoire et les temps de résolutions du schéma de Branch and Price nous proposons un nouvel algorithme de programmation dynamique (pour résoudre le sous-problème) avec une nouvelle règle de dominance en introsuisant la notion de label représentatif (un représentant de la classe d'équivalence des labels).

Nous avons procédé à des tests sur la librairie d'instances proposée par Solomon [163] afin de valider nos choix théoriques.

Les tests et les résultats sont donnés dans [82].

7.4 Conclusion sur l'ordonnancement de tâches viticoles

Nous avons résolu de manière exacte un problème de tournées de véhicules avec fenêtres de temps et réutilisation de véhicules. L'ensemble des résultats et les comparaisons avec les résultats existants se trouvent dans la thèse de Florent Hernandez [82]. Une perspective intéressante serait la caractérisation d'instances plus adaptés à la problématique de la réutilisation de véhicules.

Les instances de Solomon [163] présentent certaines limitations du fait de leur classification selon leur topologie et non selon l'aspect temporel.

CHAPITRE

8 Conclusion et perspectives

8.1 Conclusion

Comme nous avons vu dans ce manuscrit, les problèmes d'ordonnancement sont variés et différents.

En effet, nous avons présenté plusieurs résultats de complexité et d'approximation sur divers modèles d'ordonnancement. Nous avons apporté notre modeste contribution à la classification des problèmes d'ordonnancement dans un domaine très riche en résultats. Nous avons utilisé des outils classiques

- théorie de la complexité
  - réductions polynomiales plus ou moins sophistiquées,
  - et la technique du saut qui permet de séparer les instances positives des instances négatives
- et théorie de l'approximation
  - relaxation des contraintes d'intégrité d'un programme linéaire en nombres entiers,
  - utilisation d'algorithmes polynomiaux classiques (recherche d'un couplage maximum, recherche d'une clique maximale, ...) comme sous-problème.

  Pour finir, des techniques propres aux problèmes d'ordonnancement sont utilisées
  - création d'un ordre total via un tri des tâches selon un critère et raisonnement par l'absurde pour prouver le caractère optimal d'un ordonnancement en échangeant deux éléments consécutifs de cet ordre total.

Il est important de noter que les problèmes d'ordonnancement sur les modèles présentés lié au parallélisme, reste dans la classe $\mathcal{APX}$ et que les techniques restent valides (ceci fut vérifié dans les chapitres 5 et 6) avec plus ou moins d'adaptation.

Avec les modèles présentés dans les chapitres 4, 5 et 6, nous savons que le `modèle à délai de communications hiérarchiques` est une généralisation du `modèle à délai de communications homogènes`, de même que le `modèle d'ordonnancement homogène avec délai de communications locales`. La figure 8.1 présente une vision globale des modèles présentés.

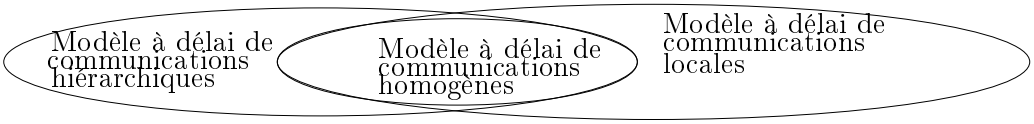


Fig. 8.1 – *Vision globale des modèles présentés liés au parallélisme*

Dans ce manuscrit, nous avons proposé plusieurs modèles d'ordonnancement qui prennent en compte divers types de communications (homogènes, hiérarchiques, locales, ...). Nous proposons ci-dessous un modèle basé sur une fonction de communications qui permet une généralisation du modèle homogène.

Pour chaque arc $(i,j) \in E$ du graphe de précédence nous associons une fonction $f(i,j,\mathcal{M}^i,\mathcal{M}^j)$ et telles que

- si $\mathcal{M}^i = \mathcal{M}^j$ alors $t_i + p_i \leq t_j$ i.e. $f(i,j,\mathcal{M}^i,\mathcal{M}^i)$ est négligeable
- sinon $\mathcal{M}^i \neq \mathcal{M}^j$ alors $t_i + p_i + f(i,j,\mathcal{M}^i,\mathcal{M}^j) \leq t_j$

où $\mathcal{M}^i$ est la machine (dans le sens général (cluster/processeur)) sur laquelle i s'exécute.

Avec cette approche, nous retrouvons les modèles à communications étudiés dans ce manuscrit :

- si $f(i,j,\pi^i,\pi^j) = c_{ij}$ avec $\pi^i \neq \pi^j$, nous retrouvons le `modèle d'ordonnancement à délai de communications homogènes` ;
- si $f(i,j,\pi^k,\pi^l) = c_{ij}d(\pi^k,\pi^l)$ avec $\pi^l \neq \pi^k$, nous retrouvons le `modèle d'ordonnancement à délai communications locales` ;
- si $f(i,j,\mathcal{M}^i,\mathcal{M}^j) = (c_{ij},\epsilon_{ij})$[1] et $\mathcal{M}^i = (\Pi^i,\pi^i)$ (resp. $\mathcal{M}^j = (\Pi^j,\pi^j)$) et $\mathcal{M}^i \neq \mathcal{M}^j$, nous retrouvons le `modèle d'ordonnancement à délai de communications hiérarchiques` ;

Ainsi, nous pourrions tenter d'étudier un modèle à délai de communications généralisées qui engloberaient les divers modèles à communication (voir la figure 8.2 pour un exemple de hiérarchies des modèles). Ce modèle devra prendre en compte plusieurs niveaux de communications entre les processeurs, machines, clusters et également des diverses topologies.

[1] Les différentes valeurs de la fonction de communications sont données au Chapitre 5.

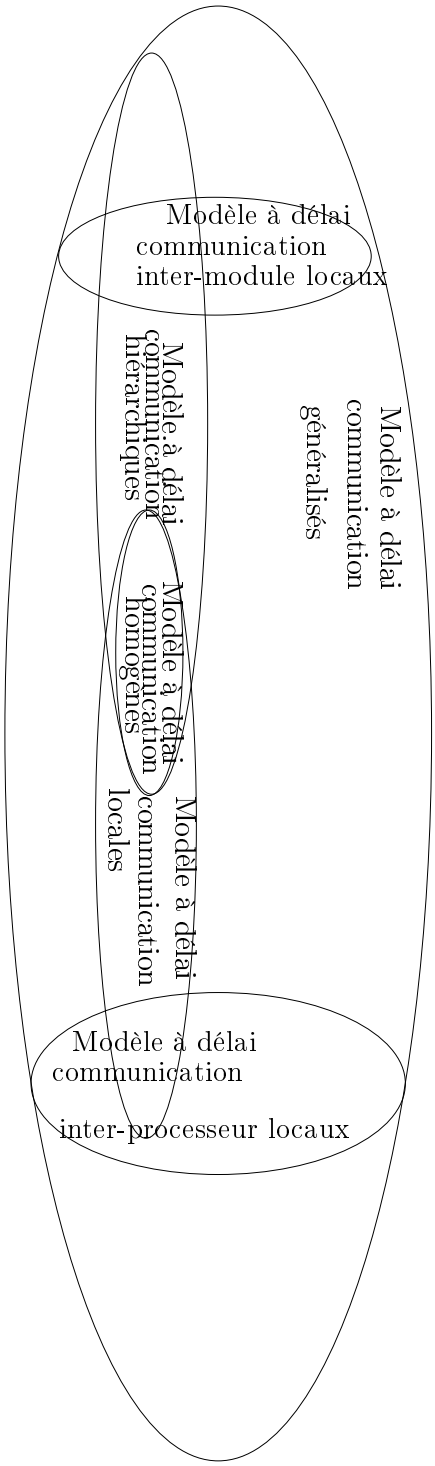


Fig. 8.2 – *A la recherche d'un modèle à délai de communication généralisée*

Nous avons également abordé des problèmes d'ordonnancement dans un autre cadre que celui du parallélisme. Nous avons étendu le modèle des tâches-couplées avec la prise en compte de la notion d'incompatibilité. La plupart des problèmes restent dans la classe $\mathcal{APX}$. Cependant certains ratios de performance dépendent des données de l'instance ; ceci apparaît quand la valeur du paramètre L_i (communication exacte) est une constante sans relation avec les autres paramètres. Nous pouvons noter que l'absence de contraintes chronologiques remplacées par des contraintes de compatibilité rend les problèmes plus difficiles au sens de la complexité. Le problème général d'ordonnancement avec des contraintes de compatibilité est non-approximable à rapport constant (la nature des problèmes se rapproche de la détermination d'une coloration des sommets avec le minimum de couleurs, même pour des variantes simples).

Pour finir nous avons utilisé des algorithmes très simples en ordonnancement pour résoudre un problème de tournées de véhicules avec limitation du nombre de véhicules. Nous aurions pu utiliser des algorihtmes spécifiques (spécialement dans le cas d'un véhicule, en procédant à l'analogie un processeur équivaut à un véhicule) mais nous avons privilégié des stratégies génériques indépendantes du nombre de véhicules/processeurs.

En conclusion, les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire pour lesquels le modèle analytique (tel qu'il est présenté) est manipulable, car décrit simplement. Ainsi, il est possible d'étudier ces problèmes via une approche théorique (classification au sens de la complexité des problèmes ; $\mathcal{NP}$-complétude, développement d'algorithmes polynomiaux efficaces, développement d'algorithmes d'approximation avec garantie de performance non triviale, méthodes exactes ...).

8.2 Perspectives

8.2.1 Etude des problèmes d'ordonnancement avec l'approximation différentielle

8.2.1.1 Les limites de l'approximation polynomiale classique

Dans cette thèse, nous nous sommes focalisés sur la classification des problèmes au sens de la théorie de la complexité et de la théorie de l'approximation avec la mesure classique basée sur le rapport entre la solution générée par notre heuristique et la solution optimale. Cette mesure n'est pas stable concernant le passage au complémentaire. En effet, il est facile de constater

que le problème de la recherche d'un stable de cardinal maximum et le problème de la recherche d'une couverture minimale des arêtes par les sommets possèdent des liens très étroits. Le complémentaire de tout stable maximum est une couverture et réciproquement, ce qui rend ces problèmes très proches et équivalents du point de vue de l'optimisation (maximiser la taille d'un stable est équivalent à minimiser celle de son complémentaire). Pourtant, ces deux problèmes admettent des comportements opposés du point de vue de l'approximation classique puisque le stable maximum compte parmi les problèmes les plus difficiles à approcher alors que l'on connaît des algorithmes 2-approchés pour la couverture. Plus généralement, la mesure classique d'approximation crée une dissymétrie entre les problèmes de maximisation et de minimisation. Elle est également très sensible à certaines opérations élémentaires (translation, changement d'origine, ...)

Au lieu de se référer simplement au pire des cas pour un algorithme approché, deux points de repères seront utilisées : l'optimum, mais aussi la pire des solutions (nous verrons que la notion de pire des solutions en ordonnancement en général doit être adaptée). Nous présenterons une extension possible de cette notion pour le problème d'ordonnancement.

8.2.1.2 Présentation de la mesure différentielle

Pour une instance fixée I, elle mesure la position de la valeur approchée $A(I)$ entre la meilleure valeur $opt(I)$ et la pire valeur $\omega(I)$. La valeur $A(I)$ qui se réfère à la solution de l'algorithme A sur une instance I est comprise entre $[opt(I), \omega(I)]$.

Le principe de l'approximation différentielle, est naturellement de se rapprocher au maximum de $opt(I)$ ou de s'éloigner au maximum de $\omega(I)$. Ainsi, on ne compare pas la valeur de la solution à celle de l'optimum, mais le chemin déjà parcouru depuis la pire solution à l'étendue des valeurs possibles, donnée par le diamètre $diam(I) = |opt(I) - \omega(I)|$. La qualité d'une solution approchée ρ^A_{diff} pour un algorithme A, en approximation différentielle, est donc donnée par la valeur du rapport

$$\rho^A_{diff} = \min_I \frac{|\omega(I) - A(I)|}{|\omega(I) - opt(I)|}$$

Cette mesure est intéressante et les résultats existants en approximation sont parfois à l'opposé des résultats d'approximation classique. Nous pouvons citer par exemple le problème de la coloration de sommets (un des problèmes de base dans l'optimisation combinatoire dans lequel le but est de colorier (avec le minimum de couleurs) les sommets du graphe tel que

deux sommets reliés par une arête ne peuvent avoir la même couleur. Dans la théorie de l'approximation avec la mesure classique, ce problème est non-approximable c'est à dire qu'il n'existe pas d'algorithme qui garanti un ratio à valeur constant (on peut s'éloigner tant qu'on veut de la solution optimale). En revanche, dans la cas de l'utilisation de la mesure différentielle il existe un algorithme ayant un ratio à facteur constant (en fait il existe un algorithme 1/2-approché et un autre à facteur 2/3 [128]). Notons la stabilité de la mesure différentielle, à la différence de la mesure classique les rapports différentiels pour le problème du stable de taille maximum et d'un transversal de taille minimum coïncide.

La mesure différentielle semble bien définie pour les problèmes d'optimisation combinatoire pour lesquels le pire des cas est facile à caractériser. Pour les problèmes d'ordonnancement le pire des cas n'a pas beaucoup de sens, il faut donc restreindre cette notion.

8.2.1.3 Application aux problèmes d'ordonnancement

Le pire des cas pour les problèmes d'ordonnancement consiste à mettre les tâches à exécuter le plus loin possible de $t = 0$ où t désigne le début de l'ordonnancement. Ainsi, la pire solution $\omega(I)$ n'est pas bornée (ceci est vrai pour tout problème d'optimisation combinatoire dont la pire solution n'est pas bornée). Ainsi, lorsque que nous considérons les problèmes d'ordonnancement nous n'allons pas considérer la pire des solutions possibles mais la pire des solutions donnée par un algorithme générique et glouton.

L'algorithme générique et glouton de base en ordonnancement est l'algorithme de liste qui consiste en une liste de tâches (triée ou non) à exécuter sur un ensemble de processeurs tel que nous prenons à chaque instant où un processeur π est disponible la première tâche disponible i dans la liste, et nous exécutons i sur π. Cette définition nous semble la plus judicieuse et la plus adaptée à l'étude des problèmes d'ordonnancement. Nous obtenons ainsi un ordonnancement glouton de durée $f(x)$ où x est une tâche de la liste des tâches. La valeur de la pire des solutions serait donc $\omega(I) = \max_x f(x)$ (remarquons que $opt(I) \neq \min_x f(x)$).

Dans ce cadre, il serait intéressant de comparer les résultats des deux mesures afin de vérifier la stabilité ou non selon la mesure des résultats d'approximation pour les problèmes d'ordonnancement.

8.2.1.4 Motivation de l'introduction de la mesure différentielle pour les problèmes d'ordonnancement

Tout le long de ce manuscrit, nous avons focalisé notre étude sur des problèmes d'ordonnancement off-line c'est-à-dire les données sont connues à l'avance. Cependant, une autre branche de l'ordonnancement traite des problèmes où les tâches arrivent à la volée et elles doivent être affectées à des processeurs. Les principes des algorithmes portent sur la création d'une liste de priorités. Dans ce cas là, nous mesurons le ratio de compétitivité. Rares sont les algorithmes qui fonctionnent efficacement dans le cas on-line et off-line. Nous pouvons citer l'algorithme de liste de Graham développé pour le problème $P||C_{max}$ qui admet un ratio dont la valeur est égale pour le cas off-line et on-line.

La mesure différentielle permettrait de relier ces deux branches de l'ordonnancement. En effet, le pire des cas pourrait être vu comme une approche on-line via une liste de priorité, et nous comparerons les résultats par rapport à un algorithme off-line. Ainsi, nous pourrons étalonner nos algorithmes off-line par rapport à l'algorithme générique pour les problèmes on-line.

8.2.2 Avec l'approche FTP (Fixed-Parameter Algorithms)

Dans le but de mieux comprendre la difficulté de résolution des problèmes, nous pourrions étudier les problèmes d'ordonnancement avec communications avec l'approche complexité paramétrique. Cependant, la plupart des résultats obtenus en complexité paramétrique portent sur des problèmes de graphes. A notre connaissance peu de résultats ont été obtenus en ordonnancement ([66], [63], [41]).

La complexité paramétrique permet de proposer des bornes inférieures opérationnelles pour des problèmes $\mathcal{NP}$-complets. La possibilité d'exhiber un algorithme FPT permet d'obtenir des algorithmes efficaces sous certaines conditions car ils limitent l'explosion combinatoire à un paramètre, et lorsqu'il est de petite taille la complexité devient « raisonnable ». La caractéristique fondamentale est la notion de noyau. Le fait d'obtenir un noyau polynomial permet, en temps polynomial, de diminuer la taille de l'instance initiale et ainsi de se ramener à un problème de taille inférieure. Un peu plus formellement, un algorithme FPT est un algorithme qui résout un problème $\mathcal{P} = (n, k)$ en temps $p(n)*f(k)$ où p est un polynôme dont les coefficients sont indépendants de k et f est une fonction quelconque. Pour un problème $\mathcal{NP}$-complet, l'algorithme de résolution est obligatoirement exponentiel (sauf si

$\mathcal{NP} = \mathcal{P}$) en la taille de l'instance, donc $f(k)$ n'est pas polynomial en k. Un problème est FPT (sous-entendu en k) s'il existe un algorithme FPT pour le résoudre.

L'étude paramétrique pourrait se focaliser sur divers paramètres comme, sur le nombre de machines, sur les caractéristiques des tâches et/ou bien sur la durée de communication ou introduire des paramètres structurels.

Nous pouvons trouver des analogies et une sorte d'équivalence entre les deux théories ;

- la classe FPT joue sensiblement le même rôle que la classe $\mathcal{P}$,
- la classe $W[1]$ joue le même rôle que la classe $\mathcal{NP}$,
- et pour finir la classe $W[1]$-complet joue le même rôle que la classe $\mathcal{NP}$-complet.

Nous pouvons noter qu'il existe une hiérarchie plus conséquente que celle de la théorie de la complexité classique ; $FTP \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[Sat] \subseteq W[P] \subseteq XP$ (Pour plus de détails, nous référençons les deux livres suivants sur la complexité paramétrique [121] et [66]).

8.2.3 Trois approches méthodologiques complémentaires

Une autre perspective intéressante consiste à évaluer, comparer et de mutualiser l'information fournie par trois approches de résolution de problèmes d'optimisation combinatoires très différentes : la résolution exacte, la résolution approchée avec garantie de performance, la résolution approchée sans garantie de performance.

Les méthodes de résolution exactes fournissent, par définition, la solution optimale d'un problème d'optimisation combinatoire. Malheureusement, la théorie de la complexité est pessimiste quant à la possibilité d'utiliser de manière efficace de telles méthodes. Il est communément admis aujourd'hui que tout problème $\mathcal{NP}$-difficile, c'est-à-dire l'essentiel des problèmes d'optimisation combinatoire sur lesquels se concentrent les recherches, ne peut être résolu de manière exacte qu'avec un algorithme de complexité exponentielle. En pratique, malgré tout, de nombreuses instances réalistes de problèmes difficiles ont pu être résolues exactement en temps raisonnable.

Les méthodes de résolution approchées sans garantie de performance sont des méthodes heuristiques permettant de s'approcher de la solution optimale avec un temps de calcul raisonnable et sont essentiellement utilisées comme palliatif des méthodes exactes lorsque ces dernières ne peuvent être utilisées efficacement. Le principal critère dans le développement d'une telle méthode est sa faculté à trouver une solution de valeur proche de l'optimum. Les

avancées récentes ont permis le développement de méthodes extrêmement efficaces, grâce notamment au développement des meta-heuristiques. Pourtant de telles méthodes ont l'inconvénient majeur de ne pas offrir de garantie quant au résultat obtenu : des comparaisons avec des bornes peuvent généralement confirmer a posteriori le bon comportement de l'heuristique, mais aucune garantie n'existe a priori.

Les méthodes de résolution approchée avec garantie de performance sont des méthodes heuristiques ayant l'avantage par rapport aux méthodes précédentes de garantir un écart maximal entre la valeur de la solution optimale et celle de la solution dans le pire des cas fournie par l'algorithme. En contrepartie, les solutions fournies sont généralement peu exploitables en pratique, car de qualité moyenne nettement inférieure à celles fournies par des heuristiques sans garantie de performance.

Ces perspectives trouvent leurs origines dans les interactions possibles entre les trois types d'approches évoquées ci-dessus, dans l'objectif d'aider à la résolution efficace de problèmes d'optimisation combinatoire. Les pistes de réflexion à explorer sont nombreuses. Deux pistes préliminaires peuvent servir de points de démarrage au projet.

La première piste est issue de la remarque suivante. La classification usuelle entre problèmes $\mathcal{NP}$-difficiles et problèmes polynomiaux permet de séparer les problèmes en deux grandes catégories. Une troisième catégorie apparaît lorsque l'on différencie les problèmes $\mathcal{NP}$-difficiles au sens fort et au sens faible (i.e., selon qu'il existe un algorithme de résolution exacte de complexité pseudo-polynomiale ou non). Malgré tout, au sein d'une même catégorie, l'expérience peut montrer des comportements très différents pour les méthodes de résolution exacte. Les résultats théoriques sur l'existence d'algorithmes approchés avec garantie de performance, ainsi que la qualité d'approximation de ces algorithmes, semblent pouvoir donner des informations supplémentaires sur la difficulté de résolution des problèmes. Nous allons tenter d'établir des liens entre la difficulté de résolution des problèmes et les classes de complexité définies par ces résultats théoriques. Nous allons également tenter de voir si ces résultats théoriques fournissent des informations sur la méthodologie de résolution exacte de ces problèmes. Des pistes équivalentes peuvent être envisagées pour une résolution heuristique sans garantie de performance au lieu d'une méthode exacte.

La deuxième piste concerne la réduction du fossé existant entre méthodes dédiées à un bon comportement en moyenne (heuristiques sans garantie de performance) et méthodes dédiées à un bon comportement dans le pire des cas (heuristiques avec garantie de performance). Par ce projet nous désirons tendre vers le développement de méthodes garantissant un bon comporte-

ment simultanément en moyenne et dans le pire des cas. Ceci passe par une analyse préliminaire du comportement moyen des heuristiques avec garantie de performance et du comportement dans le pire des cas des heuristiques sans garantie de performance. Pour cela il existe des techniques probabilistes (voir [65] pour une large présentation des techniques).

En dernier lieu nous pouvons étudier l'influence de la solution initiale sur la vitesse de convergence et sur la qualité de solution. En effet, il serait intéressant de mesurer l'impact d'une solution initiale obtenue par une algorithme d'appproximation avec garantie de performance par rapport à une solution initiale choisie de manière aléatoire.

Bibliographie

[1] F. Afrati, E. Bampis, L. Finta, and I. Milis. Scheduling trees with large communications on two processors. In A. Bode et al. (Eds.), editor, *EuroPar'00 Parallel Processing*, volume 1900 of *Lecture Notes in Computer Science*, pages 288–295. Springer-Verlag, 2000.

[2] A. A. Ageev and A. V. Kononov. Approximation algorithms for scheduling problems with exact delays. In *WAOA*, pages 1–14, 2006.

[3] A.A. Ageev and A.E. Baburin. Approximation algorithms for uet scheduling problems with exact delays. *Operations Research Letters*, 35(4) :533–540, 2007.

[4] Alexander A. Ageev and Alexander V. Kononov. Approximation algorithms for scheduling problems with exact delays. In *WAOA*, pages 1–14, 2006.

[5] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. *Annals of Mathematics*, 160 :781–793, 2004.

[6] 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, 2004.

[7] J.M. Van Den Akker, J.A. Hoogeveen, and S.L. Van De Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6) :862–872, November-December 1999.

[8] H.H. Ali and H. El-Rewini. An optimal algorithm for scheduling interval ordered tasks with communication on n processors. *Journal of Computing and System Sciences*, 51 :301–306, 1995.

[9] T.E. Anderson, D.E. Culler, D.A. Patterson, and the NOW team. A case for NOW (networks of workstations). *IEEE Micro*, 15 :54–64, 1995.

[10] T. Andronikos, N. Koziris, G. Papakonstantinou, and P. Tsanakas. Optimal Scheduling for UET/UET-UCT Generalized n-Dimensional

Grid Task Graphs. *Journal of Parallel and Distributed Computing*, 57 :140–165, 1999.

[11] E. Angel, E. Bampis, and R. Giroudeau. Non-approximability results for the hierarchical communication problem with a bounded number of clusters. In R. Feldman B. Monien, editor, *EuroPar'02 Parallel Processing*, LNCS, No. 2400, pages 217–224. Springer-Verlag, 2002.

[12] S. Arora, C. Lund, R. Motwani ans M. Sudan, and M. Szegedy. Proof verification and intractibility of approximation problems. In *Proc. 33rd IEEE Symp. on Fundations of Computer Science*, pages 13–22, 1992.

[13] G. Ausiello, P. Crescenti, and M. Protasi. Approximation solution of $\mathcal{NP}$ optimization problems. *Theoritical Computer Science*, pages 150–155, 1995.

[14] G. Ausiello and M. Protasi. Local search, reducibility and approximability of $\mathcal{NP}$-optimization problems. *Information Processing Letters*, 54 :73–79, 1995.

[15] G. Aussiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*, chapter 3, pages 100–102. Springer, 1999.

[16] J. Błażewicz, K. Ecker, , E. Pesch, G. Schmidt, and J. Węglarz. *Handbook on Scheduling*. Springer, 2007.

[17] J. Błażewicz, K. Ecker, T. Kis, and M Tanaś. A note on the complexity of scheduling coupled tasks on a single processor. *Discrete Mathematics*, 1999.

[18] J. Błażewicz, K. Ecker, G. Schmidt, and J. Węglarz. *Scheduling in Computer and Manufacturing Systems*. Springer-Verlag, 1993.

[19] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 1996.

[20] N. Azi, M. Gendreau, and J. Y. Potvin. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, 178(3) :755–766, 2007.

[21] N Azi, M Gendreau, and J. Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3) :756–763, May 2010.

[22] E. Bampis. The complexity of short-schedules for UET bipartite graphs. *RAIRO Operations Research*, 33(3) :367–370, 1997.

[23] E. Bampis, C. Delorme, and J.C. König. Optimal Schedule for d-D Grid Graphs with Communication Delays. *Parallel Computing*, 24(11) :1653–1664, 1998.

[24] E. Bampis, A. Giannakos, and J.C. König. On the complexity of scheduling with large communication delays. *European Journal of Operation Research*, 94 :252–260, 1996.

[25] E. Bampis, R. Giroudeau, and J.-C. König. Using duplication for the precedence constrained multiprocessor scheduling problem with hierarchical communications. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *EuroPar'99 Parallel Processing*, Lecture Notes in Computer Science, No. 1685, pages 369–372. Springer-Verlag, 1999.

[26] E. Bampis, R. Giroudeau, and J.-C. König. Using duplication for the precedence constrained multiprocessor scheduling problem with hierarchical communications. In P. Amestoy et al., editor, *EuroPar'99 Parallel Processing*, LNCS, No. 1685, pages 369–372. Springer-Verlag, 1999, an extended version to appear in PPL.

[27] E. Bampis, R. Giroudeau, and J.-C. König. A heuristic for the precedence constrained multiprocessor scheduling problem with hierarchical communications. In H. Reichel and S. Tison, editors, *Proceedings of STACS*, LNCS No. 1770, pages 443–454. Springer-Verlag, 2000.

[28] E. Bampis, R. Giroudeau, and J.C. König. Some results on scheduling in the presence of hierarchical communications. In *Journées de l'informatique Messine*, Mai 1999.

[29] E. Bampis, R. Giroudeau, and J.C. König. Using duplication for multiprocessor scheduling problem with hierarchical communications. *Parallel Processing Letters*, 10(1) :133–140, 2000.

[30] E. Bampis, R. Giroudeau, and J.C. König. On the hardness of approximating the precedence constrained multiprocessor scheduling problem with hierarchical communications. *RAIRO-RO*, 36(1) :21–36, 2002.

[31] E. Bampis, R. Giroudeau, and J.C. König. An approximation algorithm for the precedence constrained scheduling problem with hierarchical communications. *Theoretical Computer Science*, 290(3) :1883–1895, January 2003.

[32] E. Bampis, R. Giroudeau, and J.C. König. L'impact des communications hiérarchiques sur les problèmes d'ordonnancement. In *Renpar'11*, pages 97–102, Rennes, juin 1999.

[33] E. Bampis, R. Giroudeau, and J.C. König. Approximation results for the precedence constrained multiprocessors scheduling. In *International Conference on Optimization*. Trier (Allemagne), mars 1999.

[34] E. Bampis, R. Giroudeau, and A. Kononov. How to schedule precedence constrained tasks with small hierarchical communication delays. In *MAPSP'01, Fifth Workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 24–25, 2001.

[35] E. Bampis, R. Giroudeau, and A. Kononov. Scheduling tasks with small communication delays for clusters of processors. *Annals of Operations Research*, 1(129) :47–63, 2004.

[36] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transaction Parallel Distributed Systems*, (4), 2004.

[37] R.E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 1(16) :87–90, 1958.

[38] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg. On optimal strategies for cycle-stealing in networks of workstations. *IEEE Trans. Comp.*, 46 :545–557, 1997.

[39] E. Blayo, L. Debreu, G. Mounie, and D. Trystram. Dynamic loab balancing for ocean circulation model adaptive meshing. In P. Amestoy et al., editor, *Proceedings of Europar*, LNCS No. 1685, pages 303–312. Springer-Verlag, 1999.

[40] R. Blumafe and D.S. Park. Scheduling on networks of workstations. In *3d Inter Symp. of High Performance Distr. Computing*, pages 96–105, 1994.

[41] H.L. Bodlaender and M.R. Fellows. w[2]-hardness pf precedence constrained k-processor scheduling. Technical Report UU-CS-1994-14, Utrecht University, 1994.

[42] V. Boudet, Y. Cohen, R. Giroudeau, and J.C. Konig. Scheduling in the presence of processor networks : complexity and approximation. *RAIRO-RO*, 46(1) :1–22, 2012.

[43] M. Bouznif. Ordonnancement et communications locales. Master's thesis, Université de Montpellier II, 2008.

[44] M. Bouznif and R. Giroudeau. Inapproximability results for scheduling problem on arbitrary processors networks. *Advances in Operations Research*, 2011.

[45] R. Brent. The parallel evaluation of general arithmetic expression. *Journal of the ACM*, 21(2) :201–206, 1974.

[46] F. Cappello, P. Fraignaud, B. Mans, and A. L. Rosenberg. HiHCoHP-Towards a Realistic Communication Model for Hierarchical HyperClusters of Heterogeneous Processors, 2001. Proceedings of IPDPS'01,IEEE/ACM,IEEE Press.

[47] I. Charon, A. Germa, and O. Hudry. *Méthodes d'optilisation combinatoire*. Masson, 1996.

[48] B. Chen, C.N. Potts, and G.J. Woeginger. A review of machine scheduling : complexity, algorithms and approximability. Technical Report Woe-29, TU Graz, 1998.

[49] P. Chrétienne and J.Y. Colin. C.P.M. scheduling with small interprocessor communication delays. *Operations Research*, 39(3) :680–684, 1991.

[50] P. Chrétienne and C. Picouleau. *Scheduling Theory and its Applications*. John Wiley & Sons, 1995. Scheduling with Communication Delays : A Survey, Chapter 4.

[51] Ph. Chrétienne. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *EUR. J. Op. Res.*, 43 :225–230, 1989.

[52] F.A. Chudak and D.S. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25(5) :199–204, December 1999.

[53] E.G. Coffman and R.L. Graham. Optimal scheduling for two processors systems. *Acta Informatica*, 1 :200–213, 1972.

[54] S. Darbha and D.P. Agrawal. Optimal Scheduling Algorithm for Distributed-Memory Machines. *IEEE Transactions on Parallel and Distributed Systems*, 9(1) :87–94, January 1998.

[55] W. Fernandez de la Wega and G. Lueker. Bin packing can be solved in $1 + \epsilon$ in linear time. *Combinatorica*, 1(4) :349–355, 1981.

[56] D. de Werra. *Éléments de programmation linéaire avec application aux graphes*. Presses polytechniques romandes, 1990.

[57] T. Decker and W. Krandick. Parallel real root isolation using the descartes method. In *HiPC99*, volume 1745 of *LNCS*. Sringer-Verlag, 1999.

[58] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14(4) :545–565, 1984.

[59] P.F Dutot and D. Trystram. Scheduling on hierarchical clusters using malleable tasks. In *13th ACM Symposium of Parallel Algorithms and Architecture*, pages 199–208, 2001.

[60] K.H. Ecker and H. Hodam. Heuristic algorithms for the task scheduling under consideration of communication delays. Technical report, T.U. Clausthal, 1996.

[61] H. El-Rewini and T.G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel Distributing Computing*, 9(2) :138–153, 1990.

[62] H. El-Rewini, T.G. Lewis, and H.H. Ali. *Task scheduling in parallel and distributed system*. Prentice Hall Series in Innovative Technology, 1994.

[63] M. R. Fellows and C. McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Sciences*, 2(298) :317–324, 2003.

[64] L. Finta, Z. Liu, I. Milis, and E. Bampis. Optimal schedules for UET-UCT series parallel digraphs on two processors. *Theoretical Computer Science*, 162 :323–340, 1996.

[65] F. Flajolet and R. Sedgewick. *An introduction to the Analysis of Algorithms*. Addison-Wesley, 1995.

[66] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science ; An Eatcs Series. 2006.

[67] M.R. Garey and D.S. Johnson. *Computers and Intractability, a Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, 1979.

[68] R. Giroudeau. *L'impact des délais de communications hiérarchiques sur la complexité et l'approximation des problèmes d'ordonnancement*. PhD thesis, Université d'Évry Val d'Essonne, 2000.

[69] R. Giroudeau. Seuil d'approximation pour un problème d'ordonnancement en présence de communications hiérarchiques. *Technique et Science Informatique*, 24(1) :95–124, 2005.

[70] R. Giroudeau. Seuil d'approximation pour le modèle uet-uct en présence d'une infinité de processeurs : une preuve alternative. *Technique et Science Informatique*, 27(5) :571–588, 2008.

[71] R. Giroudeau and J.C. König. General non-approximability results in presence of hierarchical communications. In *Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pages 312–319. IEEE, 2004.

[72] R. Giroudeau and J.C. König. General scheduling non-approximability results in presence of hierarchical communications. *European Journal of Operational Research*, 184(2) :441–457, Jan 2008.

[73] R. Giroudeau, J.C. König, F.K. Moulaï, and J. Palaysi. Complexity and approximation for the precedence constrained scheduling problem with large communication delays. *Theoretical Computer Science*, 401(1–3) :107–119, 2008.

[74] R. Giroudeau, J.C. König, F.K. Moulaï, and J. Palaysi. Complexity and approximation for the precedence constrained scheduling problem with large communications delays. In P.D. Medeiros J.C. Cunha, editor, *Proceedings of Europar*, LNCS, No. 3648, pages 252–261. Springer-Verlag, 2005.

[75] R. Giroudeau, J.C. König, and B. Valéry. Scheduling uet-tasks on a star network : complexity and approximation. *4OR A Quarterly Journal of Operations Research*, 9(1) :29–48, 2011.

[76] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, 45 :1563–1581, 1966.

[77] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory : a survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[78] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Algorithms and Combinatorics,Springer,Berlin, 1988.

[79] F. Guinand, C. Rapine, and D. Trystram. Worst case analysis of lawler's algorithm for scheduling trees with communication delays. *IEEE Transaction on Parallel and Distributed Systems*, 8(10), 1997.

[80] F. Guinand and D. Trystram. Optimal scheduling of UECT trees on two processors. *RAIRO Operations Research*, 34(2) :131–144, 2000.

[81] F. Hernandez. Model-checking et ordonnancement : applications à la décision de protection phytosanitaire de la vigne. Master's thesis, Université de Montpellier II, 2007.

[82] F. Hernandez. *Méthodes de résolution exactes pour le problème de routage de véhicules avec fenêtre de temps et routes multipes.* PhD thesis, Université de Montpellier II, 2010.

[83] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud. Multi-trip vehicule routing problem with time windows for agricultural tasks. In *Odysseus, International Workshop on Freight Transportation and Logistics*, 2009.

[84] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud. An exact method to solve the multi-trip vehicle routing problem with time windows and limited duration. In *Seventh Triennial Symposium Transportation Analysis*, pages 366–369, 2010.

[85] F. Hernandez, D. Feillet, R. Giroudeau, O. Naud, and J.C. König. Problème de tournées de véhicules avec routes multiples pour réaliser des traitements phytosanitaires. In *ROADEF*, pages 5–6, 2009.

[86] D.S. Hochbaum. *Approximation Algorithm for NP-hard Problems*. PWS Publishing Company, 1997.

[87] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems : Theoritical and pratical results. *JACM*, 1(34) :144–162, January 1987.

[88] H. Hoogeveen, P. Schuurman, and G.J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In R.E. Bixby, E.A. Boyd, and R.Z. Ríos-Mercado, editors, *IPCO VI*, Lecture Notes in Computer Science, No. 1412, pages 353–366. Springer-Verlag, 1998.

[89] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16(3) :129–137, 1994.

[90] I. Hoyler. The NP-completeness of edge-coloring. *SIAM Journal of Computing*, 10 :718–720, 1981.

[91] J.-J. Hwang, Y.C. Chow, F.D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing*, 18 :244–257, 1989.

[92] A. Jakoby and R. Reischuk. The complexity of scheduling problems with communication delays for trees. In *Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory*, pages 165–177, 1992.

[93] H. Jung, L.M. Kirousis, and P. Spirakis. Lowers bounds and efficient algorithms for multiprocessor scheduling of directed acyclic graphs with communication delays. *Inf. Comput.*, 105 :94–104, 1993.

[94] D. Jungnickel. *Graphs, Networks and algorithms*, volume 5 of *Algorithms and Compution in Mathematics*. Springer, 2002.

[95] Y. Kopidakis. *Approximabilité des problèmes d'ordonnancement dans les systèmes parallèles*. PhD thesis, Université de Paris-Sud, 1996.

[96] C. Lahlou. Scheduling with unit processing and communication times on a ring network : Approximation results. In *Proceedings of Europar*, pages 539–542. Springer-Verlag, 1996.

[97] E.L. Lawler. Scheduling trees on multiprocessors with unit communication delays. In *Worshop on Models and algorithms for Planning and Scheduling Problems*, pages 14–18, Villa Vigoni, Lake Como, Italy, June 1993.

[98] J.K. Lenstra and Rinnooy Kan A.H.G. Complexity of scheduling under precedence constraints. *Operation Research*, 26, 1978.

[99] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Operations Research*, 1 :343–362, 1977.

[100] J.K. Lenstra and A.H.G. Rinnoy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26 :22–35, 1978.

[101] J.K. Lenstra, M. Veldhorst, and B. Veltman. The complexity of scheduling trees with communication delays. *Journal of Algorithms*, 20 :157–173, 1996.

[102] W. T. Ludwig. *Algorithms for scheduling malleable and nonmalleable parallel tasks.* PhD thesis, University of Wisconsin-Madison, Department of Computer Sciences, 1995.

[103] R.H. Möhring. *Computationnally tractable classes of ordered sets.* Kluwer Academic, Dordrecht, 1989.

[104] R.H. Möhring and M.W. Schäffter. Scheduling series-parallel orders subject to 0/1-communication delays. *Parallel Computing*, 25(1) :23–40, January 1999.

[105] R.H. Möhring, M.W. Schäffter, and A.S. Schulz. Scheduling jobs with communication delays-using solutions for approximation. In *Proceedings of the Fourth European Symposium on Algorithms*, volume 1136 of *Lecture Notes in Computer Science*, pages 76–90. Springer-Verlag, September 1996.

[106] A. Moukrim. Optimal scheduling on parallel machines for a new order class. *Operations Research Letters*, 24 :91–95, 1999.

[107] G. Mounié. *Efficient scheduling of parallel application : the monotic malleable tasks.* PhD thesis, Institut National Polytechnique de Grenoble, 2000.

[108] G. Mounié, C. Rapine, and D. Trystram. Efficient approximation algorithm for scheduling malleable tasks. In *11th ACM Symposium of Parallel Algorithms and Architecture*, pages 23–32, 1999.

[109] A. Munier. *Algorithmes approchés our les problèmes d'ordonnancement à temps de communication.* PhD thesis, Université Pierre et Marie Curie, 1996.

[110] A. Munier. Approximation algorithms for scheduling trees with general communication delays. *Parallel Computing*, 25(1) :41–48, 1999.

[111] A. Munier and C. Hanen. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. In *IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, 1995.

[112] A. Munier and C. Hanen. An approximation algorithm for scheduling unitary tasks on m processors with communication delays. Private communication, 1996.

[113] A. Munier and C. Hanen. Using duplication for scheduling unitary tasks on m processors with communication delays. *Theoretical Computer Science*, 178 :119–127, 1997.

[114] A. Munier and C. Hanen. Performance of Coffman-Graham schedules in the presence of unit communcation delays. *Discrete Applied Mathematics*, 81 :93–108, 1998.

[115] A. Munier and J.C. König. A heuristic for a scheduling problem with communication delays. *Operations Research*, 45(1) :145–148, 1997.

[116] A. Munier, M. Queyranne, and A.S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *IPCO*, Lecture Notes in Computer Science, No. 1412. Springer-Verlag, 1998.

[117] O. Naud, T. Tuitete, B. Léger, A. Hélias, and R. Giroudeau. Système à événements discrets : de la simulation à l'analyse temporelle de la décision en agriculture. In *5ème journées STIC et Environnement 2007*, 2007.

[118] O. Naud, T. Tuitete, B. Léger, A. Hélias, and R. Giroudeau. Systèmes réactifs pour modéliser la décision en production agricole. In *Modélisation des systèmes réactifs (MSR)*, pages 159–174, 2007.

[119] O. Naud, T. Tuitete, B. Léger, A. Hélias, and R. Giroudeau. Système à événements discrets : de la simulation à l'analyse temporelle de la décision en agriculture. *Sciences et Technologie de l'Automatique*, 5(2), 2008. Special STIC & Evironnement.

[120] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[121] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications 31, 2006.

[122] A.J. Orman and C.N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72 :141–154, 1997.

[123] M.A. Palis, J.C. Liou, and D.S.L. Wei. Task Clustering and Scheduling for Distributed Memory Parallel Architectures. *IEEE Transactions on Parallel and Distributed Systems*, 7(1) :46–55, January 1996.

[124] C.H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[125] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Prentice Hall, 1982.

[126] C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comp.*, 19(2) :322–328, April 1990.

[127] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity. *Journal of computer and System Sciences*, 43 :425–440, 1991.

[128] V. Paschos. *Complexité et approximation polynomiale.* Hermès-Science Publication, 2004.

[129] G.F. Pfister. *In Search of Clusters.* Prentice-Hall, 1995.

[130] C. Picouleau. $UET - UCT$ schedules on arbitrary networks. Technical report, LITP, Blaise Pascal, Université Paris VI, 1994.

[131] C. Picouleau. New complexity results on scheduling with small communication delays. *Discrete Applied Mathematics*, 60 :331–342, 1995.

[132] C. Picouleau. *Ordonnancement pour l'informatique parallèle*, chapter Ordonnancement sur deux processeurs avec contraintes de capacité pour la transmission des communications, pages 41–50. Hermès, 2003.

[133] M. Pinedo. *Scheduling : theory, Algorithms, and Systems.* Prentice Hall, 1995.

[134] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58 :263–285, 1993.

[135] M. Queyranne and A.S. Schulz. Polyhedral Approaches to Machine Scheduling. Technical Report Preprint 408/1994, Technical University of Berlin, Departement of Mathematics, 1994. Berlin.

[136] M. Queyranne and A.S. Schulz. Scheduling unit jobs with compatible release dates on parallel machines with nonstationary speeds. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization*, number 920 in Lecture Notes in Computer Science, pages 307–320. Proceedings of the 4th International IPCO Conference, 1995.

[137] M. Queyranne and Y. Wang. A cutting plane procedure for precedence-constrained single machine scheduling. Technical report, Faculty of Commerce, University of British Columbia,Canada, 1991. Working paper.

[138] M. Queyranne and Y. Wang. Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research*, 16 :1–20, 1991.

[139] V. Raman, B. Ravikumar, and S. Srinivasa Rao. A simplified $\mathcal{NP}$-complete MAXSAT problem. *Information Processing Letters*, 65(1) :1–6, 15 January 1998.

[140] C. Rapine. *Algorithmes d'approximation garantie pour l'ordonnancement de tâches, Application au domaine du calcul parallèle.* PhD thesis, Institut National Polytechnique de Grenoble, 1999.

[141] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18 :55–71, 1987.

[142] A.L. Rosenberg. Guidelines for data-parallel cycle-stealing in networks of workstations I : on maximizing expected output. *Journal of Parallel Distributing Computing*, 59(1) :31–53, 1999.

[143] A.L. Rosenberg. Guidelines for data-parallel cycle-stealing in networks of workstations II : on maximizing guarantee output. *Intl. J. Foundations of Comp. Science*, 11 :183–204, 2000.

[144] F. Rossi, P. Van Beek, and T. Walsh. *Hanbook of Constraint programming.* Elsevier, 2006.

[145] R. Saad. Scheduling with communication delays. *JCMCC*, 18 :214–224, 1995.

[146] M.W. Schäffter. *Scheduling Jobs with Communication Delays, Complexity Results and Approximation Algorithms.* PhD thesis, Technical University of Berlin, 1996.

[147] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1998.

[148] A.S. Schulz. *Polytopes and Scheduling.* PhD thesis, Technical University of Berlin, 1995.

[149] A.S. Schulz. Scheduling to minimize total weighted completion time : performance guarantees of LP-based heuristics and lower bounds. In W.H. Cunningham, S.T. McCormick, and Q. Queyranne, editors, *Interger Programming and Combinatorial Optimization*, number 1084, pages 301–315. Proceedings of the 5th International IPCO Conference, 1996.

[150] R.D. Shapiro. Scheduling coupled-tasks. *Naval Research Logistics Quaterly*, 20 :489–498, 1980.

[151] P. Siarry, J. Dréo, A. Pétrowski, and E. Taillard. *Métaheuristique pour l'optimisation difficile*. Eyrolles, 2003.

[152] G. Simonin. *L'impact de l'introduction du graphe de compatibilité dans les problèmes d'ordonnancement en présence de tâches-couplées*. PhD thesis, Université de Montpellier II, 2009.

[153] G. Simonin, R. Giroudeau andJ.C. König, and B. Darties. Theoretical aspects of scheduling coupled-tasks in the presence of compatibility graph. In *ICAPS*, pages 218–225, 2011.

[154] G. Simonin, R. Giroudeau andJ.C. König, and B. Darties. Theoretical aspects of scheduling coupled-tasks in the presence of compatibility graph. *Algorithmic in Operations Research*, 7(1) :1–12, 2012.

[155] G. Simonin, B. Darties, R. Giroudeau, and J.C. König. Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor. In *Fourth Multidisciplinary International Scheduling Conference, MISTA'09*, pages 378–388, 2009.

[156] G. Simonin, B. Darties, R. Giroudeau, and J.C. König. Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor. *Journal of Scheduling*, 14(5) :501–509, 2011.

[157] G. Simonin, R. Giroudeau, and J.C. König. Complexity and approximation for scheduling problem for a torpedo. In *The 39th International Conference on Computers and Industrial Engineering, IEEE*, pages 300–304, 2009.

[158] G. Simonin, R. Giroudeau, and J.C. König. Complexity and approximation for scheduling problem for coupled-tasks in presence of compatibility tasks. In *Project Management and Scheduling*, 2010.

[159] G. Simonin, R. Giroudeau, and J.C. König. Polynomial-time algorithms for scheduling problem for coupled-tasks in presence of treatment tasks. *Electronic Notes in Discrete Mathematics*, 36 :647–654, 2010.

[160] G. Simonin, R. Giroudeau, and J.C. König. Polynomial-time algorithms for scheduling problem for coupled-tasks in presence of treatment tasks. In *International Symposium on Combinatorial Optimization*, 2010,.

[161] G. Simonin, R. Giroudeau, and J.C. König. Complexity and approximation for scheduling problem for a torpedo. *Computers & Industrial Engineering*, 61(2) :352–356, 2011.

[162] G. Simonin, R.Giroudeau, and J.-C. König. Extended matching problem for a coupled-tasks scheduling problem. In *TMFCS :International Conference on Theoretical and Mathematica Foundations of Computer Science, Orlando Florida*, pages 82–89, 2009.

[163] M.M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35 :254–265, 1987.

[164] R. Thurimella and Y. Yesha. A scheduling principle for precedence graphs with communication delay. In *International Conference on Parallel Processing*, volume 3, pages 229–236, 1992.

[165] Groupes thématiques de PRS Capa-Rumeur-Exec. *ICARE'97,Conception et mise en øeuvre d'applications parallèles irrégulières de grandes tailles*, chapter 5, page 117. CNRS, décembre 1997.

[166] V. T'Kindt and J.C. Billaut. *Multicriteria Scheduling : Theory, Models And Algorithms*. Springer, 2006.

[167] T. Tuitete. Ordonnancement de tâches phytosanitaires viticoles. Master's thesis, Université de Montpellier II, 2006.

[168] J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In *4th ACM Symposium of Parallel Algorithms and Architecture*, pages 323–332, 1992.

[169] B. Valéry. Prise en compte de la topologie pour les problèmes d'ordonnancement. Master's thesis, Université de Montpellier II, 2006.

[170] T.A. Varvarigou, V.P. Roychowdhury, T. Kailath, and E.L. Lawler. Scheduling in and out forests in the presence of communication delays. *IEEE Transactiona on Parallel and Distributed System*, 7 :1065–1074, 1996.

[171] V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[172] B. Veltman. *Multiprocessor scheduling with communication delays*. PhD thesis, CWI-Amsterdam, Holland, 1993.

[173] B. Veltman, B. J. Lageweg, and J.K. Lenstra. Multiprocessor scheduling with communications delays. *Parallel Computing*, 16 :173–182, 1990.

[174] J. Verriet. Scheduling interval-ordered tasks with non-uniform deadlines subject to non-zero communication delays. *Parallel*, 25(1) :3–21, January 1999.

[175] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A. J. Hurkens, J.K. Lenstra, S.V. Sevast'janov, and D.B. Shmoys. Short shop schedules. *Operations Research*, 45(2) :288–294, March–April 1997.