

ACADÉMIE DE MONTPELLIER
UNIVERSITÉ MONTPELLIER II
— SCIENCES ET TECHNIQUES DU LANGUEDOC —

MÉMOIRE DE STAGE DE MASTER

SPÉCIALITÉ : **Recherche en Informatique**
Mention : **Informatique, Mathématiques, Statistiques**

effectué au laboratoire LIRMM/INFO

—
sous la direction de RODOLPHE GIROUDEAU

**Complexité et Approximation pour les problèmes
d'ordonnancement avec contraintes de localité**

par

Benoît VALÉRY

soutenu le 19 Juin 2006

Résumé : Dans ce document, nous proposons un nouveau model d'ordonancement multi-processeur dans lequel les processeurs ne sont plus tous reliés les uns aux autres. Nous introduisons un graphe de processeurs qui représente les liens entre chaque processeur. Le délai de communication entre deux processeurs dépend maintenant de la distance des deux processeurs dans ce graphe, ce qui oblige a bien choisir quel processeur exécutera quelle tâches. Nous étudierons plusieurs dispositions différentes pour le graphe des processeurs (étoile, grille, tore, . . .). Pour chacun de ces graphes des processeurs nous montrerons la complexité du problème d'ordonancement associé. Nous donnerons aussi deux algorithmes avec garantie de performance lorsque le graphe de processeur est une étoile.

Abstract : In this document, we give a new scheduling model where all processors aren't link to each others. We add a new graph, named network of processor, that represents the relation between processors. The communication delay between two processors now depends on the distance between this two processors in the network. In this way, you have to choose carefully which processor will perform a given task. We study here some differents network of processor (star, mesh, tore, . . .). For each network we give the complexity of the corresponding scheduling problem. We also give two approximations algorithms when the network of processor is a star.

Table des matières

1	Introduction	5
1.1	Présentation du modèle	6
1.2	Notions de bases	7
1.2.1	Définitions	7
1.2.2	Notations des problèmes	9
1.3	État de l'art	10
1.3.1	Modèles sans communication	10
1.3.2	Modèles avec communications	10
1.3.3	Modèle avec prise en compte de la topologie du réseau des processeurs	11
1.4	Présentation du rapport	11
2	L'étoile	12
2.1	Introduction	12
2.2	Présentation de l'étoile et notations	12
2.2.1	Le graphe	12
2.2.2	Du point de vue de l'ordonnancement	13
2.3	Complexité lorsque les valuations sont unitaires	14
2.3.1	Seuil d'approximation pour un graphe quelconque et pour la minimisation de la longueur de l'ordonnancement	14
2.3.2	Seuil d'approximation pour un graphe quelconque et pour la somme des temps de complétude	19
2.4	Algorithmes avec garanties de performances	21
2.4.1	Détails des étapes	21
2.4.2	Algorithmes	23
2.5	Généralistaion des résultats	24
2.5.1	Complexité	24
2.5.2	Approximation	25
2.6	Conclusion	25
3	La Grille	26
3.1	Introduction	26
3.2	Présentation de la grille	26
3.3	Ordonnancement sur une grille/tore à m processeurs	27
3.3.1	La grille	27
3.3.2	Le tore	32

3.4	La distance entre deux processeurs n'est plus unitaire	32
3.5	Conclusion	33
4	L'arbre binaire équilibré	34
4.1	Introduction	34
4.2	Présentation de l'arbre binaire équilibré	34
4.3	Ordonnancement sur un arbre binaire équilibré à m processeurs .	34
4.4	Conclusion	37
5	Conclusion	38
5.1	Conclusion	38
5.2	Perspectives	39

Chapitre 1

Introduction

Le traitement informatique de grandes applications est de plus en plus fréquent pour éviter des simulations réelles qui sont souvent très coûteuses (décollage de fusée) ou dangereuses pour l'environnement (armement militaire). L'utilisation d'une machine grand public pour effectuer les calculs séquentiellement mettrait beaucoup trop de temps pour pouvoir donner les résultats souhaités.

Ce besoin a permis l'apparition de Supercomputer dans les années 60 proposé par *Seymour Cray*. Seulement ces machines, souvent gigantesques, sont très peu pratiques. En effet, il fallait souvent une vingtaine de personnes qualifiées pour son utilisation et la machine était aussi grande qu'un bâtiment.

Si ces machines, à cause de leurs inconvénients (prix, maintenance...), sont réservés qu'à très peu de monde, elles ont permis de prouver l'utilité du parallélisme (utilisation de plusieurs processeurs). La recherche s'est naturellement tournée vers les calculs distribués afin de permettre l'utilisation d'un grand nombre de machine grand public comme groupe de travail.

À partir d'une grandes applications jusqu'à sa résolution sur plusieurs machines, nous rencontrons beaucoup de problèmes. D'une part, il faut être capable de découper l'application de départ en plusieurs applications plus petites qui seront réparties sur les différentes machines. Cette étape transforme l'application de départ en une application dites parallèle représentée par un graphe orienté acyclique (graphe de précédence). Chaque sommet de ce graphe représente un ensemble d'instructions et les arcs représentent une contrainte de précédence (c'est-à-dire la tâche à l'extrémité de l'arc doit s'exécuter au plus tôt après la fin de l'exécution de la tâches à l'origine de l'arc).

Après cette étape, il faut assigner les différents ensembles d'instructions sur les différentes machines en respectant les contraintes de précédence. Cette étape est l'ordonnancement à proprement parler et nous étudierons cette partie dans ce rapport. Beaucoup de modèles d'ordonnancement ont été étudié au fil des années. Tout d'abord, les communications entre deux tâches ont été considéré comme instantanées. Ensuite, un délai de communication a été ajouté lorsque deux tâches s'exécutent sur deux processeurs différents ([1] et [2]). Dans ce modèle, chaque processeur est relié à tous les autres, par conséquent la localisation d'exécution des tâches n'influe pas sur la valeur du délai de communication,

dans ce cas le réseau est dit homogène.

Dans ce rapport nous proposons un modèle qui prend en compte le placement des tâches sur les différentes machines. Tous les processeurs ne sont plus reliés à tous les autres, d'où l'importance du placement des tâches. En effet, un processeur communiquera plus vite avec un processeur voisin plutôt qu'un processeur loin de lui.

1.1 Présentation du modèle

Une instance de notre problème d'ordonnancement est donné par :

- un ensemble $V = \{1, \dots, n\}$ de n tâches non préemptives (dont on ne peut suspendre l'exécution),
- un ensemble E de contraintes de précédence sur (i, j) tel que $G = (V, E)$ est un graphe orienté acyclique,
- une durée d'exécution $p_i \forall i \in V$,
- un ensemble $V^* = \{1, \dots, m\}$ de m processeurs équivalents en puissance,
- un ensemble E^* une relation sur $V^* \times V^*$ tel que $G^* = (V^*, E^*)$ est un graphe non orienté connexe.
- Le délai de communication entre deux tâches i et j soumises à une contrainte de précédence (i, j) est égale à la distance $d(\pi^i, \pi^j)$ qui sépare les processeurs. Formellement nous avons :

$$\forall (i, j) \in E, t_j \geq t_i + p_i + d(\pi^i, \pi^j)$$

où :

1. π^i désigne le processeur qui exécute i ,
2. t_i désigne le début d'exécution de i ,
3. p_i désigne le temps d'exécution de i ,
4. $d(\pi^i, \pi^j)$ est le plus court chemin entre le processeur qui exécute i et celui qui exécute j sur le graphe des processeurs.

Tout au long de ce rapport nous utiliserons des durées de tâches unitaires et la distance entre deux processeurs voisins sera soit unitaire pour tous couples de processeurs voisins soit égale à $\alpha \in \mathbb{N}$ pour tous couples de processeurs voisins.

Exemple

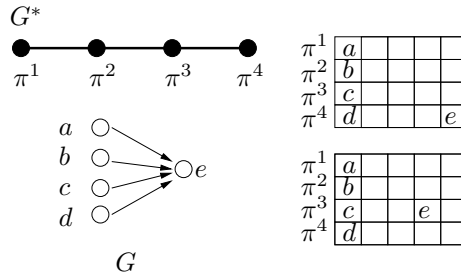


FIG. 1.1 – Exemple d'ordonnancement du modèle avec contraintes de localité

Dans cet exemple, nous voyons que la fin d'exécution de la tâche e dépend de son placement. Si la tâche e est exécutée sur les processeurs aux extrémités du graphe des processeurs G^* (π_1 et π_4), nous perdons une unité de temps par rapport au cas où la tâche e est exécutée sur un des processeurs au centre de la chaîne.

1.2 Notions de bases

1.2.1 Définitions

Définition 1.1 (Graphe de précédence) *Un graphe de précédence $G = (V, E)$ est un graphe orienté acyclique (parfois valué) où :*

- un sommet représente une tâche ou un ensemble d'instructions,
- un arc représente une relation de précédence entre ces tâches,
- la valuation sur arc (i, j) représente le délai de communication potentiel entre la fin d'exécution de i et le début d'exécution de j .

Définition 1.2 (Graphe des processeurs) *Un graphe de processeurs $G^* = (V^*, E^*)$ est un graphe non orienté valué où :*

- un sommet représente un processeur,
- une arête représente une liaison physique directe entre les deux processeurs,
- une valuation sur une arête représente la distance qui sépare les deux processeurs.

Définition 1.3 (Délai de communication) *Un délai de communication, noté c_{ij} , entre deux tâches i et j est le temps d'attente minimum pour exécuter j après la fin d'exécution de i si les deux tâches sont exécutées sur deux processeurs différents.*

Notation 1.4 *Soit i une tâche (un sommet de G) et π un processeur (un sommet de G^*), nous noterons par π^i si le processeur π exécute la tâche i .*

Notation 1.5 *Pour toute instance I d'un problème d'ordonnancement, nous noterons par $C_{max}^h(I)$ la longueur de l'ordonnancement obtenu par l'heuristique h sur l'instance I . De même $C_{max}^{opt}(I)$ désignera la solution optimale sur I .*

Définition 1.6 (Algorithme Approché) *Soit A un algorithme d'ordonnancement. On dit que l'algorithme A est ρ -approché si nous avons pour toute instance I :*

$$\rho \leq \max \frac{C_{max}^h(I)}{C_{max}^{opt}(I)}$$

Définition 1.7 (Duplication) *La duplication permet d'avoir plusieurs instances d'une même tâche i .*

La duplication a été introduite par Papadimitriou et Yannakakis afin de réduire l'influence des délais de communication ([3]).

S'il existe un algorithme approché pour le problème d'ordonnancement avec une garantie de performance strictement plus petite que $\frac{b}{a}$, alors cet algorithme pourrait séparer les instances positives et les instances négatives du problème d'ordonnancement, fournissant ainsi un algorithme polynomial pour un problème \mathcal{NP} -Complet.

Par conséquent, le problème d'ordonnancement ne possède pas d'algorithme ρ -approché, avec $\rho < \frac{b}{a}$.

1.2.2 Notations des problèmes

Etant donné le grand nombre de paramètres qui peuvent varier d'un modèle à l'autre, Graham ([4]) et Blazewicz et al. ([5]) ont proposé une notation pour les problèmes d'ordonnancement. Nous allons donner cette notation et l'étendre pour notre modèle.

Un problème d'ordonnancement est décrit par trois paramètres α, β, γ . Le détail de ces trois paramètres est le suivant :

- $\alpha = \{\alpha_1, \alpha_2\}$ décrit le graphe des processeurs.
 - $\alpha_1 = \{P, \bar{P}, P_m\}$ donne le nombre de processeurs et α_2 donne la topologie du graphe des processeurs.
 - $\alpha_1 = P$ signifie que le nombre de processeurs m est une entrée du problème.
 - $\alpha_1 = \bar{P}$ signifie que nous avons une infinité de processeurs.
 - $\alpha_1 = P_m$ le nombre de processeurs est fixé par m .
 - $\alpha_2 = \{K_n, \cdot\}$ signifie que le graphe des processeurs est un graphe complet, tous les processeurs sont reliés entre eux.
 - $\alpha_2 = \{\text{Grille}\}$ signifie que le graphe des processeurs est sous forme de grille.
- $\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ décrit le graphe de précédence où
 - $\beta_1 = \{\text{prec}, \text{arbre}, \text{biparti}, \dots\}$ donne la topologie du graphe de précédence (quelconque, un arbre, ...).
 - $\beta_2 = \{\cdot, 1, c_{ij}, d_{G^*}(\pi^i, \pi^j)\}$ donne les délais de communication (sans communication, coût unitaire ou coût qui dépend du placement des tâches i et j).
 - $\beta_3 = \{1, p_i\}$ donne les durées d'exécution des tâches (unitaires, quelconque).
 - $\beta_4 = \{\cdot, \text{dup}\}$ traduit si les tâches peuvent être dupliquées (i.e. il peut y avoir plusieurs exécutions d'une même tâche).
 - $\beta_5 = \{\cdot, \text{pmtp}\}$ dit si on permet l'interruption d'une tâche en cours d'exécution.
- γ est la fonction objectif du problème. Le plus souvent nous utilisons C_{max} qui cherche à minimiser la longueur totale de l'ordonnancement. Plus rarement nous avons $\sum C_j$ (où $C_j = t_j + p_j$ avec t_j le date de début d'exécution de j) qui cherche à minimiser la somme des temps de complétude de l'ensemble des tâches.

Exemple de notations de problèmes

$\overline{P} \mid \text{prec}, c_{ij} = 1, p_i = 1 \mid C_{max}$ est le problème qui cherche à ordonnancer sur une infinité de processeurs un graphe de précédence quelconque dont les tâches et les délais de communication sont unitaires et dont la fonction objective est la minimisation de la longueur de l'ordonnancement. Ce modèle est souvent appelé modèle UET-UCT (Unit Execution Time - Unit Communication Time) ([1]).

$\{P, \text{étoile}\} \mid \text{prec}, c_{ij} = d_{G^*}(\pi^i, \pi^j), p_i = 1 \mid C_{max}$ est le problème qui cherche à minimiser la longueur de l'ordonnancement sur un nombre de processeur donné en paramètre placé en étoile. Le graphe de précédence est quelconque, la durée d'exécution des tâches est unitaire et le délai de communication potentiel dépend de la distance qui sépare les deux processeurs.

1.3 État de l'art

1.3.1 Modèles sans communication

Au début de l'ordonnancement, les problèmes étudiés ne prenaient en compte que peu de paramètres. Tout d'abord les tâches étaient toutes indépendantes (le graphe de précédence n'est composé que de tâches isolées). Mais ces problèmes restaient difficiles. Par exemple, nous avons en autres les problèmes :

- $P \parallel C_{max}$ qui est équivalent au problème *BinPacking* ([6]). Un algorithme de liste ([4]) pour ce problème donne une 2-approximation quelque soit la liste. Trier les tâches de la liste par ordre décroissant de durée d'exécution améliore la borne à $\frac{4}{3} - \frac{1}{3m}$ -approximation.
- $P_2 \parallel C_{max}$ qui est équivalent au problème *Partition* ([6]).

Ensuite, la topologie du graphe de précédence est devenue une donnée importante dans un problème d'ordonnancement. Nous pouvons citer par exemple les problèmes suivant :

- $P \mid \text{prec}, p_j = 1 \mid C_{max} = 3$ est \mathcal{NP} -Complet. Un algorithme de liste donne ici une $2 - \frac{1}{m}$ -approximation.
- $P_2 \mid \text{prec}, p_j = 1 \mid C_{max}$ est polynomial.
- $P_3 \mid \text{prec}, p_j = 1 \mid C_{max}$ est ouvert.

1.3.2 Modèles avec communications

Ensuite, les modèles ont pris en compte les délais de communications lorsque deux tâches sont exécutées sur deux processeurs différents. Ceci a permis de d'étudier de nombreux nouveaux problèmes tels que :

- $\overline{P} \mid \text{prec}, p_j = 1, c_{ij} = 1 \mid C_{max} = 5$ est \mathcal{NP} -Complet. Ce problème, désigné par UET-UCT ([1]), a été très largement étudié. Une formulation du problème en programme linéaire en nombres entiers donne une $\frac{4}{3}$ -approximation.

- P | $prec, p_j = 1, c_{ij} = c$ | $C_{max} = c + 3$ est \mathcal{NP} -Complet. L'utilisation de la dilatation (détaillée dans le Chapitre 2) permet de donner une $\frac{2(c+1)}{3}$ -approximation.
- P_2 | $prec, p_j = 1, c_{ij} = 1$ | C_{max} est ouvert.

1.3.3 Modèle avec prise en compte de la topologie du réseau des processeurs

L'article [7] a introduit ce modèle. Cet article étudie le cas où le graphe des processeurs est une chaîne ou un cycle et quand la distance entre deux processeurs voisins est unitaire. La chaîne a été choisie comme première topologie pour sa forme. La chaîne a une structure régulière et ses sommets sont de degrés deux à l'exception des deux extrémités.

L'article donne une borne inférieure d'approximabilité lorsque le graphe de précedence est quelconque ou biparti.

Leurs résultats peuvent être repris par le tableau ci-dessous.

G^*		G	Complexité		Borne d'approx.	
Topologie	$d(\pi^i, \pi^j)$		Poly.	$\mathcal{NP} - \mathcal{C}$	Inf.	Sup.
Chaîne	1	prec, biparti	2	3	4/3	
Cycle	1	prec, biparti	2	3	4/3	

1.4 Présentation du rapport

Dans le chapitre deux, nous traiterons le cas où le graphe des processeurs est sous forme d'étoile. Nous proposerons la borne inférieure d'approximabilité puis nous proposerons un algorithme approché pour ce problème.

Dans le chapitre trois, nous étudierons le cas où le graphe des processeurs est une grille, nous généraliserons ce résultat pour les tores.

Dans le chapitre quatre, nous montrerons la borne inférieure lorsque le graphe des processeurs est un arbre binaire. Ce chapitre permet d'amener la conclusion sur les différentes topologies pour les graphes de processeurs.

Chapitre 2

L'étoile

2.1 Introduction

La première topologie étudiée pour le graphe des processeurs sera l'étoile. Nous choisissons ce graphe comme point de départ du fait qu'il se rapproche d'un graphe complet. Nous présenterons d'abord les caractéristiques de ce graphe, ainsi que différentes notations utiles pour la suite du chapitre. Ensuite nous étudierons la complexité du problème pour déterminer la limite entre les problèmes faciles et difficiles. Nous travaillerons tout d'abord dans le cas où les valuations sur les arêtes sont unitaires, puis nous étendrons certains résultats pour une valeur entière. Nous proposerons ensuite un algorithme approché lorsque le problème sera difficile.

2.2 Présentation de l'étoile et notations

2.2.1 Le graphe

Définition 2.1 *Un graphe étoile $G = (V, E)$ est un arbre où tous les sommets sont des feuilles sauf un qui est voisin de tous les autres.*

Notation 2.2 *Nous noterons le sommet du centre de l'étoile par π_* et les autres par (π_2, \dots, π_n) .*

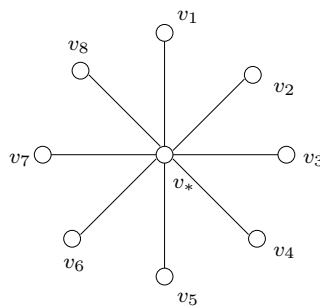


FIG. 2.1 – Une étoile à 9 sommets

Si nous tenons pas compte du sommet au centre de l'étoile, chaque sommet est à égale distance de tous les autres, c'est sur ce point là que nous pouvons le comparer à un graphe complet.

2.2.2 Du point de vue de l'ordonnancement

Lorsque nous allons ordonnancer les différentes tâches issues d'un graphe de précedence sur l'étoile, le processeur qui est au centre de l'étoile pourra communiquer avec un coût inférieur que deux processeurs extérieurs. En effet on a :

$$d(\pi_i, \pi_j) = d(\pi_i, \pi_*) + d(\pi_*, \pi_j) \text{ avec } i \neq j$$

Ceci nous amène à caractériser sur le diagramme de Gantt un des processeurs comme celui se trouvant au centre de l'étoile. Le processeur associé au sommet π_* sera dans la suite de ce chapitre celui en bas du diagramme comme montré sur la Figure ci-dessous.

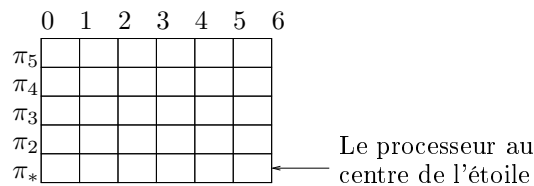


FIG. 2.2 – Le diagramme de Gantt associé à une étoile

Par conséquent, lorsque deux tâches i et j avec $(i, j) \in E$ s'exécutent sur deux processeurs différents, le placement de i et j sur les différents processeurs sera fondamental. En effet la longueur totale de l'ordonnancement pourra changer si le processeur au centre de l'étoile n'aura pas été bien utilisé.

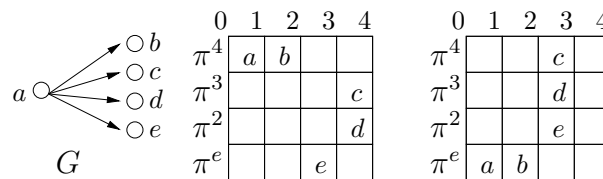


FIG. 2.3 – La longueur de l'ordonnancement des tâches de G diffère suivant l'utilisation du processeur central

Le problème étudié dans ce chapitre s'écrit formellement de la façon suivante :

$$\{P, \text{etoile}\} | prec, c_{ij} = d_{G^*}(\pi^i, \pi^j), p_i = 1 | C_{max}$$

où $d_{G^*}(\pi_i, \pi_*) = \alpha \in \mathbb{N}$

pour $i = 2, \dots, n$

C'est à dire, nous allons ordonnancer les tâches d'un graphe de précedence quelconque sur m processeurs disposés en étoile dont les arêtes sont valuées par α .

2.3 Complexité lorsque les valuations sont unitaires

2.3.1 Seuil d'approximation pour un graphe quelconque et pour la minimisation de la longueur de l'ordonnancement

Théorème 2.3 *Le problème de décider si une instance du problème $\{P, \text{étoile}\}$ prec, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1 | C_{max}$ avec $d_{G^*}(\pi^i, \pi^*) = 1$ possède un ordonnancement de longueur au plus 4 est polynomial.*

Preuve : Le graphe ne doit pas avoir plus de $4m$ tâches, sinon il n'y aurait pas assez de slots pour tout exécuter en 4 unités de temps.

Le graphe de précedence ne peut être de profondeur supérieure à 4. En effet, un chemin de longueur strictement supérieure à 4 ne pourrait pas s'exécuter en moins de 4 unités de temps, même si toutes les tâches du chemin sont exécuter consécutivement. De même tous les chemins de longueur 4 sont disjoints. Supposons le contraire, au moins l'un des deux chemins ne sera pas exécuté seulement sur le même processeur. Un délai de communication doit être pris en compte et la longueur totale de l'ordonnancement sera supérieur à 4.

À chaque instant, il ne peut y avoir plus de m tâches.

Sans prendre en compte le processeur au centre de l'étoile, les seules communications qu'il peut y avoir sont des communications entre une source et un puits. Par conséquent, les chemins de longueurs 3 doivent s'exécuter sur le même processeur (avec éventuellement un temps d'inactivité).

Une fois que le processeur au centre de l'étoile est occupé, nous pouvons très vite dire, grâce aux propriétés ci-dessus, si le graphe de précedence peut être ordonnancé en 4 unités de temps. \square

Théorème 2.4 *Le problème de décider si une instance du problème $\{P, \text{étoile}\}$ prec, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1 | C_{max}$ possède un ordonnancement de longueur au plus 5 est \mathcal{NP} -complet.*

Preuve : La démonstration est basée sur le problème de *Clique* :

Donnée : Un graphe non orienté $G = (V, E)$ et un entier k .

Question : Existe-il une clique de taille k dans G ?

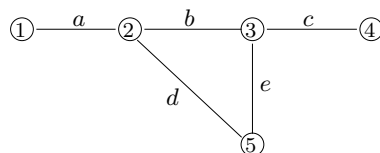


FIG. 2.4 – Une instance du problème *Clique* qui servira d'exemple pour la preuve. Les sommets 2,3,4 forment une clique de taille 3 qui est la clique maximum du graphe

On note par $l = \frac{k(k-1)}{2}$ le nombre d'arête de la clique de taille k . Nous posons par $m = 2\max\{|V| + 1, l + 1, |E| - l + 1\}$ le nombre de processeurs (nous multiplions par 2 pour éviter qu'un des ensembles définis ci-après ne soient vides).

Notre preuve est basée sur la réduction $Clique \propto \{P, \text{étoile}\} | \text{prec}$, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1 | C_{max}$. Soit une instance Φ^* du problème $Clique$, nous construisons une instance problème Φ du problème d'ordonnancement de la façon suivante :

- $\forall v \in V$ nous introduisons deux tâches sommets : T_v et K_v qui définiront respectivement les ensembles T et K .
 - $\forall e \in E$ nous introduisons une tâche arête L_e qui définira l'ensemble L .
- Nous ajoutons les contraintes de précédence suivantes : $T_v \rightarrow K_v$ et $T_v \rightarrow L_e$ si v est extrémité de e .

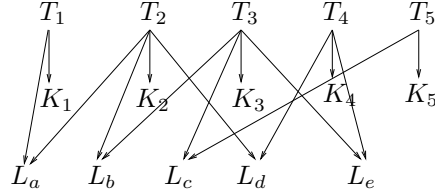


FIG. 2.5 – Illustration des précédences entre les ensembles T , K et L associé au graphe de la Figure 2.4.

Nous introduisons également 6 ensembles de tâches :

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

Sur ces ensembles nous ajoutons les contraintes de précédence suivantes :

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

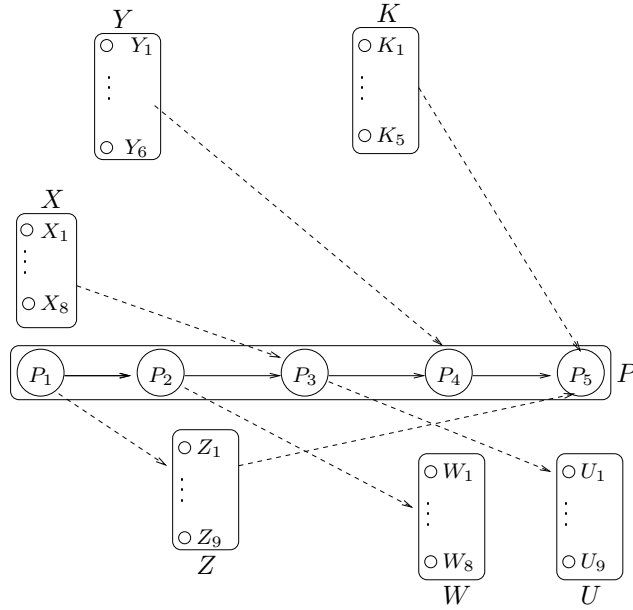


FIG. 2.6 – Illustration des précédences entre les ensembles X, Y, Z, W, U et K associé à la Figure 2.4. Une flèche en pointillée entre deux ensembles représente un graphe complet entre ces deux ensembles

La transformation se fait en temps polynomial.

- Supposons que G possède une clique de taille k , alors il existe un ordonnancement de longueur 5. Il suffit d'ordonner les tâches en se basant sur la Figure ci-dessous.

π_m	T_{Clique}	K_{Clique}	Z	L_{Clique}	$\overline{L_{Clique}}$	
\vdots	X	$\overline{T_{Clique}}$	$\overline{K_{Clique}}$	W	U	
π_2		Y	Z			
π_*	P_1	P_2	P_3	P_4	P_5	
	0	1	2	3	4	5

FIG. 2.7 – Exemple de construction d'un ordonnancement de longueur 5. T_{clique} représente les tâches de T associées aux sommets de la *clique* et $\overline{T_{clique}}$ le complémentaire.

- Réciproquement, supposons que nous avons un ordonnancement de longueur 5 alors il existe une clique de taille k . Nous allons faire des remarques essentielles sur le placement des ensembles X, Y, Z, W et U .

1. Toutes les tâches de P doivent s'exécuter sur le processeur π_* . D'une part les tâches de P forment un chemin de longueur 5 alors elles doivent s'exécuter consécutivement. D'autre part, $\forall z \in Z_z$ il existe un chemin $c = \{P_1, z, P_5\}$ puisque les tâches de P sont sur le même processeur il y a communication pour $P_1 \rightarrow z$ et $z \rightarrow P_5$. Admettons que les tâches de P ne soient pas sur π_* alors les deux communications vont prendre deux unités de temps chacune et P_5 ne pourra pas s'exécuter avant $t = 4$.
2. Toutes les tâches de X s'exécutent à $t = 0$. Elles sont prédécesseurs de P_3 (qui est exécutée à $t = 2$) donc elles ne peuvent s'exécuter qu'à $t = 0$.
3. Toutes les tâches de U s'exécutent à $t = 4$. Ces tâches sont successeurs de P_3 (qui est exécutée à $t = 2$) donc elles ne peuvent s'exécuter qu'à $t = 4$.
4. Les tâches de W s'exécutent à $t = 3$. Elles sont successeurs de P_2 (qui est exécutée à $t = 1$) donc elles ne peuvent s'exécuter qu'à $t = 3$ ou $t = 4$.
5. Les tâches de Y peuvent s'exécuter à $t = 0$ ou $t = 1$. Elles sont prédécesseurs de P_4 donc elles ne peuvent s'exécuter après $t = 1$. Lorsque nous regarderons le placement des tâches des ensembles T , K et L , nous verrons que les tâches de Y ne peuvent s'exécuter qu'à $t = 1$.
6. Les tâches de W peuvent s'exécuter à $t = 3$ ou $t = 4$. Elles sont successeurs de P_2 donc elles ne peuvent pas s'exécuter avant $t = 3$. Cependant nous verrons que toutes les tâches de W devront s'exécuter $t = 3$ lorsque nous parlerons des ensembles T , K et L .

Après le placement des ensembles ci-dessus, rappelons le nombre de slots libres à chaque instant :

- Il y a $|V| + k$ slots libres sur l'intervalle de temps $[0, 2[$. En effet, il y a deux tâches de P , $(m - k - 1)$ tâches de X et $m - |V| - 1$ tâches de Y .
- À $t = 2$ il reste $|V| - k$ processeurs libres. P_3 en utilise un et l'ensemble Z en utilise $(m - |V| + k - 1)$.
- Sur l'intervalle de temps $[3, 5[$, il y a $|E|$ slots libres. Deux slots sont occupés par deux tâches de P , $(m - l - 1)$ slots sont occupés par les tâches de l'ensemble W et $(m - |E| + l - 1)$ par les tâches de l'ensembles U .

Nous pouvons remarquer qu'il y a exactement $2|V| + |E|$ slots libres pour exécuter $2|V| + |E|$ tâches des ensembles T , K et L . Etudions maintenant le placement de ces ensembles :

1. Les tâches de K doivent s'exécuter au plus tard à $t = 2$ puisqu'elles sont prédécesseurs de P_5 . De plus elles doivent s'exécuter consécutivement par rapport à leur prédécesseur de l'ensemble T . En effet, s'il y a communication entre T_v et K_v les tâches de K_v ne pourront pas respecter leur contrainte d'exécution au plus tard à $t = 2$. De ce fait, toutes les tâches de l'ensemble K s'exécuteront soit à l'instant $t = 1$ ou $t = 2$.
2. Les tâches de l'ensemble L ont chacune deux prédécesseurs dans l'ensemble T . Comme nous pouvons déduire facilement de la remarque ci-dessus que deux tâches de T s'exécutent sur deux processeurs différents, nous devons prendre en compte le délai de communication. Soit $L_e \in L$, si les deux

prédécesseurs de L_e sont exécutés à $t = 0$ alors L_e peut s'exécuter à $t = 3$. De même si un ou les deux prédécesseurs de L_e sont exécutés à $t = 1$ alors L_e devra s'exécuter à $t = 4$. Nous avons donc que les tâches de L peuvent s'exécuter au plus tôt à $t = 3$.

3. À l'instant $t = 0$, il peut y avoir au plus k tâches (rappel : nous notons par l le nombre d'arête d'une clique à k sommets) et il ne peut y avoir moins de l slots libres à $t = 3$. Admettons qu'il y ait moins de k tâches qui s'exécutent à l'instant $t = 0$, le sous-graphe engendré par les k sommets associés à ces tâches aura forcément moins de l arêtes. Nous aurons alors des instants libres à $t = 3$ et nous ne pourrions pas avoir un ordonnancement valide en cinq unités de temps.

De même si nous avons k tâches qui s'exécutent à l'instant $t = 0$, pour avoir exactement $l = \frac{k(k-1)}{2}$ tâches pouvant s'exécuter à $t = 3$ il faut absolument que les sommets associés à ces tâches soient tous reliés entre eux, c'est à dire il faut que les k sommets forment une clique. Imaginons qu'une des tâches exécutées à l'instant $t = 0$ ne fasse pas partie de la clique, soit T_i cette tâche et T_c les $k-1$ autres. Les sommets représentés par les tâches de T_c sont reliés par $l' = \frac{(k-1)(k-2)}{2} < l$ arêtes. Le sommet représenté par la tâche T_i est relié à T_c par moins de $k-1$ arêtes, soit l'' ce nombre d'arête. Nous avons alors à l'instant $t = 3$, $l' + l'' = \frac{(k-1)(k-2)}{2} + l'' < l$ tâches exécutables, comme dans le cas précédent nous aurions des slots libres à $t = 3$ et pas assez de slots libres à $t = 4$ ce qui est impossible. Les tâches exécutés à l'instant $t = 0$ représentent des sommets faisant partis d'une même clique.

	0	1	2	3	4	5
π_{12}	T_2	K_2	Z_1	L_b	L_a	
π_{11}	T_3	K_3	Z_2	L_d	L_c	
π_{10}	T_4	K_4	Z_3	L_e	U_1	
π_9	X_1	T_1	K_1	W_1	U_2	
π_8	X_2	T_5	K_5	W_2	U_3	
π_7	X_4	Y_1	Z_4	W_3	U_4	
π_6	X_3	Y_2	Z_5	W_4	U_5	
π_5	X_5	Y_4	Z_6	W_5	U_6	
π_4	X_6	Y_3	Z_7	W_6	U_7	
π_3	X_7	Y_5	Z_8	W_7	U_8	
π_2	X_8	Y_6	Z_9	W_8	U_9	
π_*	P_1	P_2	P_3	P_4	P_5	

FIG. 2.8 – Illustration de l'ordonnancement du graphe de la Figure 2.4

□

Théorème 2.5 *Le problème de décider si une instance du problème $\{P, \text{étoile}\}$ $prec$, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1$, $dup|C_{max}$ possède un ordonnancement de longueur au plus 5 est \mathcal{NP} -complet.*

Preuve : La preuve vient directement du Théorème 2.4, du fait qu'il y a m processeurs et que le nombre de tâches est $5m$. Les tâches ne peuvent donc pas être dupliquées. \square

Corollaire 2.6 *Il n'existe pas d'algorithme approché avec une garantie de performance inférieure à $6/5$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$ pour les problèmes $\{P, \text{étoile}\}$ $prec$, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1$, $\alpha|C_{max}$ avec $\alpha \in \{., dup\}$*

Preuve : La preuve est une conséquence immédiate du Théorème de l'Impossibilité ([6] et [8]) \square

2.3.2 Seuil d'approximation pour un graphe quelconque et pour la somme des temps de complétude

Théorème 2.7 *Le problème $\{P, \text{étoile}\}$ $prec$, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1$ | $\sum C_j$ ne possède pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à $14/13$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve : Pour obtenir ce résultat, nous considérons la transformation polynomiale utilisée pour la démonstration du Théorème 2.4. Cette transformation utilise un ensemble de m machines (m est une donnée du problème) et un ensemble de tâches de durée unitaire soumises à des contraintes de précédence et des délais de communications.

Il est facile de vérifier que :

- Dans le cas où l'instance du problème *Clique* reçoit une réponse positive (c'est à dire nous pouvons conclure à l'existence d'une *clique* de taille k), dans l'instance du problème d'ordonnancement, toutes les tâches sont exécutées dans l'intervalle $[0, 5]$. Alors la somme des temps de complétude associée est $m + 2m + 3m + 4m + 5m = 15m$.
- Dans le cas où l'instance du problème reçoit une réponse négative (nous pouvons dire qu'il n'y a pas de *clique* de taille k), alors dans n'importe quel ordonnancement réalisable au moins une tâche commence son exécution après strictement $t = 4$. Comme nous cherchons une borne inférieure, nous pouvons dire qu'elle est atteinte lorsque une seule tâche s'exécute à $t = 5$ au lieu de $t = 4$. Nous avons alors $15m - 5 + 6 = 15m + 1$ (une illustration est donnée par la Figure 2.9)

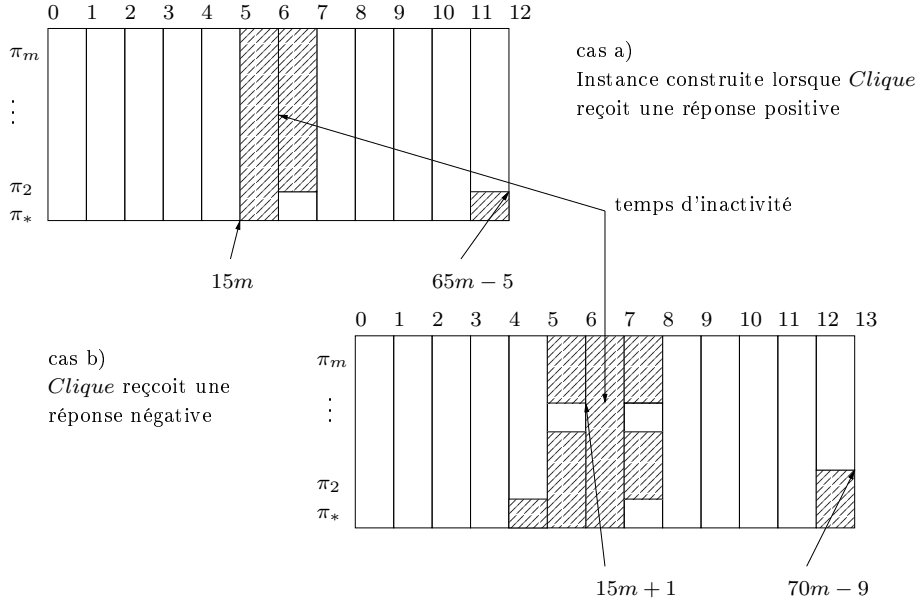


FIG. 2.9 – Construction de la transformation pour la somme des temps de complétude à partir de celle effectuée pour la longueur de l'ordonnancement

Nous ajoutons maintenant aux tâches de l'instance initiale un ensemble de $5m$ nouvelles tâches. Chaque nouvelle tâche est un successeur des anciennes tâches (les anciennes tâches sont issues de la transformation polynomiale utilisée pour la démonstration du Théorème 2.4). Nous obtenons un graphe complet entre les nouvelles tâches et les anciennes tâches.

Nous noterons cette instance I^* sur laquelle nous observons les propriétés suivante :

- Dans le cas où l'instance du problème *Clique* reçoit une réponse positive, alors à partir de l'instance I^* , nous construisons un ordonnancement dont la somme des temps de complétude est égale à $15m + 7 + 8m + 9m + 10m + 11m + 12(m - 1) = 65m - 5$ (voir la Figure 2.9 cas a)). Nous laissons deux temps d'inactivité à $t = 5$ et $t = 6$ pour assurer le coût de communication entre les anciennes tâches et les nouvelles tâches.
- Lorsque l'instance du problème *Clique* reçoit une réponse négative, alors à partir de l'instance I^* , la somme des temps de complétude est au moins égale à $15m + 1 + 16 + 9m + 10 + 11m + 12m + 13(m - 2) = 70m - 9$ (voir la Figure 2.9 cas b)). En effet au plus une tâche peut s'exécuter à $t = 7$ sur le même processeur que la tâche qui a un début d'exécution à l'instant $t = 5$ et une tâche sur π_* . Comme elle est exécutée sur le même processeur, le coût de communication est nul et nous avons bien deux unités de temps pour respecter le délai de communication entre les anciennes tâches et la tâche qui a un début d'exécution à l'instant 7.

Donc il existe un algorithme d'approximation en temps polynomial avec une garantie de performance strictement inférieure à $14/13$ qui peut être utilisé pour

distinguer en temps polynomial les instances positives des instances négatives du problème *Clique* qui est \mathcal{NP} -complet ce qui est impossible sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$. Par conséquent le problème $\{P, \text{etoile}\} | prec, c_{ij} = d_{G^*}(\pi^i, \pi^j), p_i = 1 | \sum C_j$ ne possède pas d'algorithme ρ -approché, avec $\rho > 14/13$. \square

2.4 Algorithmes avec garanties de performances

Dans cette partie nous allons proposer un algorithme approché donnant un ordonnancement lorsque le graphe de processeur est un graphe de diamètre deux (donc valable pour l'étoile). Nous allons expliquer séparément chaque partie nécessaire à l'algorithme avant d'analyser le facteur d'approximation.

2.4.1 Détails des étapes

Approximation sur le modèle UET-UCT

Notre construction d'un ordonnancement sur l'étoile sera construit à partir d'un ordonnancement sur le modèle $P_\infty | prec, c_{ij} = 1, p_i = 1 | C_{max}$. Ce problème étant aussi \mathcal{NP} -difficile, nous allons utiliser un algorithme approché sur ce modèle.

Théorème 2.8 *L'approximation sur le modèle $P_\infty | prec, c_{ij} = 1, p_i = 1 | C_{max}$ est une $(4/3)$ -approximation.*

Preuve : La preuve peut être vue dans ([9]) \square

Dilatation

Notation 2.9 *Nous noterons par σ^∞ l'ordonnancement sur une infinité de processeurs sur le modèle UET-UCT, par σ_c^∞ l'ordonnancement sur le modèle UET-LCT. De plus nous noterons par t_i (resp. t_i^c) le début d'exécution de la tâche i dans l'ordonnancement σ^∞ (resp. σ_c^∞).*

À partir d'un ordonnancement sur σ^∞ , nous récupérons les couples $\forall i \in V, (t_i, \pi^i)$ où :

- t_i est la date de début d'exécution de la tâche i ,
- π^i est le processeur sur lequel la tâche i s'exécute.

Maintenant, nous déterminons un nouveau couple $\forall i \in V, (t_i^c, \pi^{i'})$ pour l'ordonnancement sur σ_c^∞ de la façon suivante :

- le début d'exécution de la tâche i est $t_i = \frac{(c+1)}{2}t_i$
- $\pi^i = \pi^{i'}$

Les justifications peuvent être vues dans ([10]). Une illustration est donnée par la Figure 2.10.

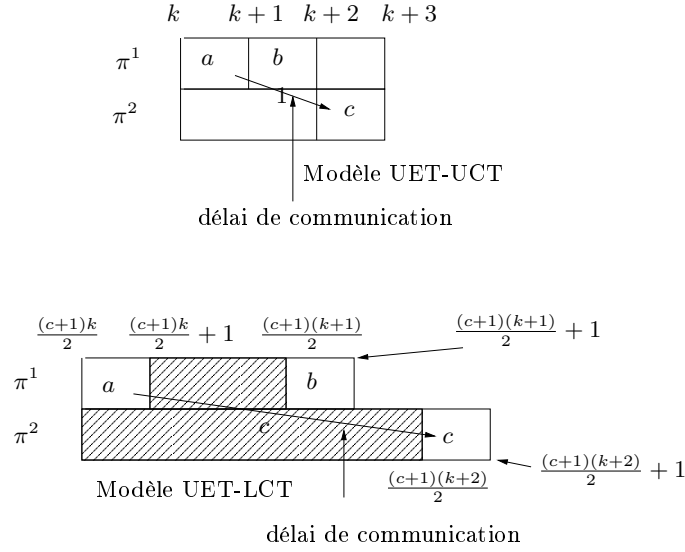


FIG. 2.10 – Illustration de la dilatation

Dans notre cas où le diamètre du graphe est deux, notre facteur de dilatation sera $3/2$.

Comparaison entre le modèle UET-LCT et un graphe de diamètre c

Dans le modèle UET-LCT toutes les communications sont égales à $c \geq 2$. Dès lors nous devons considérer le coût de communication, nous allons appliquer c unités de temps. Seulement, il se peut que sur le graphe de diamètre deux nous aurions pu effectuer la communication entre deux processeurs voisins. Il nous faut donc comparer l'écart de ces deux situations.

Théorème 2.10 *L'écart de la longueur de l'ordonnancement entre un graphe complet où les délais de communication valent c et un graphe de diamètre c est de $\frac{c+1}{2}$*

Preuve : Soit $c = \{x_1, x_2, \dots, x_n\}$ le chemin critique. Considérons qu'il y a un coût de communication entre x_i et x_{i+1} pour $1 \leq i < n$. Dans le modèle UET-LCT, la longueur de l'ordonnancement sera $(1+c)n - c$ alors que dans le cas où chaque coût de communication sur le chemin critique est unitaire, la longueur de l'ordonnancement est $2n - 1$.

Le rapport des ordonnancement sur UET-LCT et un arbre revient donc à calculer :

$$\lim_{n \rightarrow \infty} \frac{(1+c)n - c}{2n - 1} = \frac{1+c}{2}$$

□

Dans notre cas où le diamètre du graphe est deux, l'écart sera donc un facteur de $3/2$.

Pliage

Le pliage consiste à passer d'un ordonnancement sur un nombre infini de processeurs vers un nombre fini de processeurs.

Théorème 2.11 *À partir de n'importe quelle heuristique de rapport de performance ρ pour le problème $\overline{P}|prec, c_{ij} = 1, p_i = 1|C_{max}$ nous pouvons obtenir une heuristique pour le problème $P|prec, c_{ij} = 1, p_i = 1|C_{max}$ de rapport de performance $1 + \rho$.*

Le principe de cette méthode est de prendre l'ensemble X_i des tâches exécuté à l'instant i sur le modèle à une infinité de processeur et de découper cet ensemble en $\frac{|X_i|}{m}$ ensembles pour l'ordonnancement sur le modèle à m processeurs.

Les détails de la preuve peuvent être vus dans ([11])

2.4.2 Algorithmes

L'algorithme part d'un solution approché sur le modèle $\overline{P}|prec, c_{ij} = 1, p_i = 1|C_{max}$ et va la transformer pour obtenir un ordonnancement valide sur un graphe de diamètre deux. Nous avons trois opérations à appliquer pour obtenir un ordonnancement sur un arbre de diamètre deux. Deux de ces opérations sont une multiplication par $3/2$ (la dilatation et la comparaison) et une opération qui ajoute une unité de temps (le pliage).

Un premier algorithme

Cette algorithme va tout d'abord calculer un ordonnancement réalisable sur le modèle UET-UCT avec une infinité de machine. Cette solution va être ramener sur un nombre fini de machine. Nous allons ensuite modifier cette solution pour le cas où les délais de communication prennent deux unités de temps, enfin nous pourrons appliquer cet ordonnancement pour l'étoile.

Algorithme 1 : Calcul d'un ordonnancement pour l'étoile

Données : $G = (V, E)$ un graphe de précédence, m le nombre de processeur

Résultat : Un ordonnancement réalisable pour l'étoile

Calcul du début d'exécution des tâches

Calcul d'une approximation sur σ^∞

Pliage vers m processeurs

Dilatation avec $c = 2$

Construction de l'ordonnancement

pour tous les couple (t_i, π) faire

└ Placer la tâche i à l'instant t_i sur le processeur π

Le rapport d'approximation de cet algorithme est alors :

$$\rho = \left(\frac{4}{3} + 1\right) \times \frac{3}{2} \times \frac{3}{2} = \frac{21}{4}$$

Nous avons donc :

$$C_{max}^{h,etoile} \geq \frac{21}{4} C_{max}^{opt,etoile}$$

Un deuxième algorithme

L'ordre de ces opérations pour obtenir le meilleur résultat consiste à effectuer le pliage en dernier. Dans le cas contraire, le pliage ne rajoutera pas qu'une unité de temps mais 9/4 unités de temps. Nous avons donc l'algorithme suivant :

Algorithme 2 : Calcul d'un ordonnancement pour l'étoile (version 2)

Données : $G = (V, E)$ un graphe de précedence, m le nombre de processeur

Résultat : Un ordonnancement réalisable pour l'étoile

Calcul du début d'exécution des tâches

Calcul d'une approximation sur σ^∞

Dilatation avec $c = 2$

Construction de l'ordonnancement sur une étoile infinie

pour tous les couples (t_i, π) faire

└ Placer la tâche i à l'instant t_i sur le processeur π

Pliage vers m processeurs

Le rapport d'approximation de cet algorithme est alors :

$$\rho = \left(\frac{4}{3} \times \frac{3}{2} \times \frac{3}{2} \right) + 1 = 4$$

Nous avons donc :

$$C_{max}^{h,etoile} \geq 4C_m^{opt,etoile}$$

2.5 Généralistaion des résultats

Nous allons effectuer la modification suivante, les valuations sur les arêtes de l'étoile ne seront plus unitaires, mais elles seront valués par un entier α . Nous allons donner le nouveau seuil d'approximation puis nous adapterons le rapport d'approximation.

2.5.1 Complexité

Théorème 2.12 *Le problème de décider si une instance du problème $\{P, étoile\} | prec$, $c_{ij} = d_{G^*}(\pi^i, \pi^j)$, $p_i = 1 | C_{max}$ avec $d_{G^*}(\pi^*, \pi^i) = \alpha \in \mathbb{N}$ possède un ordonnancement de longueur au plus $(2\alpha + 3)$ est \mathcal{NP} -complet.*

Preuve : La démonstration est très proche de la preuve du Théorème 2.4. Nous donnerons donc la transformation du problème de *Clique* à notre problème d'ordonnancement. Le reste du raisonnement reste identique.

- $\forall v \in V$ nous introduisons une tâche T_v et un chemin de longueur α $K_v = \{K_{v_1}, K_{v_2}, \dots, K_{v_\alpha}\}$ qui définiront respectivement les ensembles T et K .
- $\forall e \in E$ nous introduisons une tâche arête L_e qui définira l'ensemble L .

Sur ces ensembles nous ajoutons les contraintes de précédence suivante :

- $\forall t \in T$ nous avons $t \rightarrow P_{\alpha+2}$,
- $\forall k \in K$ nous avons $K_{k_\alpha} \rightarrow P_{2\alpha+3}$,
- $\forall l \in L$ nous avons $P_{\alpha+1} \rightarrow l$.

Nous avons aussi des ensembles de tâches auxiliaires pour imposer un squelette aux ensembles T , K et L .

À présent, le raisonnement sur le placement des ensembles définis ci-dessus est identique à la preuve du Théorème 2.4. S'il existe une clique de taille k dans notre instance de *Clique*, les sommets de cette clique seraient les sommets associés aux tâches ordonnancées à $t = 0$ dans notre problème d'ordonnancement. \square

2.5.2 Approximation

Les outils utilisés sont les mêmes pour trouver une approximation sur un graphe de diamètre 2α . Cependant les facteurs qui interviennent lors de la dilatation et la comparaison entre un graphe de diamètre α et un graphe complet où les coûts de communication sont de α vont changer. En adaptant ces deux opérations pour des coûts de communication de α , nous obtenons :

- Le facteur de dilatation sera maintenant $\frac{\alpha+1}{2}$,
- la comparaison sera maintenant de $\frac{\alpha+1}{2}$.

Le nouveau rapport ρ d'approximation est donc :

$$\rho = \frac{4}{3} \times \frac{\alpha+1}{2} \times \frac{\alpha+1}{2} + 1$$

$$\rho = \frac{4}{3} \times \left(\frac{\alpha+1}{2}\right)^2 + 1$$

2.6 Conclusion

Dans ce chapitre nous venons d'étudier la première topologie pour le graphe des processeurs. Nous avons montré la difficulté du modèle et donné une borne inférieure d'approximation. Nous avons ensuite donné un algorithme d'approximation pour l'étoile (et pour tout graphe de diamètre deux) montrant que ce problème est approximable. Enfin nous avons généralisé ces résultats en faisant varier les valuations des arêtes de l'étoile.

Il serait possible de continuer à travailler sur l'étoile en changeant un peu le modèle. Par exemple il serait intéressant de considérer que les tâches non plus une durée unitaire ou nous pourrions mettre des valuations différentes sur chaque arête de l'étoile. Nous pourrions aussi considérer que chaque processeur est plus ou moins rapide ce qui rajouterai encore un niveau de difficulté.

Chapitre 3

La Grille

3.1 Introduction

Notre seconde topologie pour le graphe des processeurs étudiée sera la grille. La grille a un degré borné mais son diamètre dépend du nombre de processeur contrairement à l'étoile. Dans ce chapitre, nous montrerons la difficulté pour la grille lorsque le graphe de précedence est quelconque ou biparti pour la grille. Nous adapterons ensuite ces résultats lorsque le graphe des processeurs est un tore.

3.2 Présentation de la grille

Définition 3.1 Une grille à p lignes et q colonnes (noté (p, q)) est un graphe $G = (V, E)$ où :

- $|V| = pq$, et $s_{i,j}$ pour $i \in \{1, \dots, p\}$ et $j \in \{1, \dots, q\}$ représente le sommet à la i -ième ligne et j -ième colonne.
- $E = \{(x, y) | x = s_{i,j} \text{ pour } i \in \{1, \dots, p-1\} \text{ et } j \in \{1, \dots, q\}, y = s_{i+1,j}\} \cup \{(x, y) | x = s_{i,j} \text{ pour } i \in \{1, \dots, p\} \text{ et } j \in \{1, \dots, q-1\} \text{ et } y = s_{i,j+1}\}$

Notation 3.2 Nous noterons par $\pi_{i,j}$ le processeur qui est à la i -ième ligne et j -ième colonne.

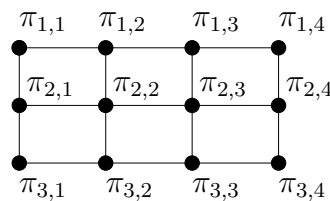


FIG. 3.1 – Un exemple de grille à 3 lignes et 4 colonnes.

Définition 3.3 Soit $H = (V', E')$ une grille (p, q) , un tore $G = (V, E)$ est une grille où :

- $V = V'$,
- $E = E' \cup \{(x, y) | x = s_{1,j} \text{ et } y = s_{p,j} \text{ pour } j \in \{1, \dots, q\}\} \cup \{(x, y) | x = s_{i,1} \text{ pour } i \in \{1, \dots, p\} \text{ et } y = s_{i,q}\}$

3.3 Ordonnancement sur une grille/tore à m processeurs

Nous allons dans un premier temps nous intéresser aux problèmes lorsque la distance entre deux processeurs voisins est unitaire. Nous ajusterons ensuite les résultats lorsque cette distance passe à $\alpha \in \mathbb{N}$.

3.3.1 La grille

Théorème 3.4 *Le problème de décider si une instance du problème $\{P, \text{grille}\}$ | $\text{prec}, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j)$ | C_{max} possède un ordonnancement de longueur 2 est polynomial.*

Preuve : Il est facile de voir si un graphe de précédence peut être ordonnancé en 2 unités de temps. En effet le graphe de précédence ne peut être composé que d'une collection de tâches isolées ou de chaînes de longueur un. \square

Théorème 3.5 *Le problème de décider si une instance du problème $\{P, \text{grille}\}$ | $\text{prec}, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j)$ | C_{max} possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

Preuve : Notre preuve est basée sur une réduction depuis le problème de *Partition*

Rappelons la définition du problème *Partition*

Données : Soit une collection d'entiers $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$ et $\forall a_i \in \mathcal{A}, v(a_i) \in \mathbb{N}$

Question : Existe-t-il un sous-ensemble $\mathcal{S} \subset \mathcal{A}$ tel que $\sum_{a \in \mathcal{S}} v(a) = \sum_{a \in \mathcal{A} \setminus \mathcal{S}} v(a)$

Pour simplifier la preuve, nous allons considérer que $\forall a \in \mathcal{A}$ on a $v(a) \geq 2$. À partir d'une instance Φ de *Partition* nous allons créer une instance Φ' de notre problème d'ordonnancement de la façon suivante :

- Le nombre $m = (p, q)$ de processeurs est égale à $(p, (4B + 1))$ où $B = \frac{\sum_{a \in \mathcal{A}} a}{2}$ et $p \in \mathbb{N}^*$.
- Nous construisons un squelette où les processeurs $\pi^{i,j}$ pour $i \geq 2$ et $j \in \{1, \dots, 4B + 1\}$ sont saturés par des tâches tandis que la première ligne de processeurs possède des slots libres. Un exemple de squelette pour une grille à deux lignes de processeurs est décrit par la Figure 3.2. Les parties notées $P[i, j]$ sont des chemins de longueur trois qui occuperont les processeurs $\pi^{i,j}$ tandis que les tâches notées $T[i, j]$ occuperont seulement un slot du processeur $\pi^{i,j}$.

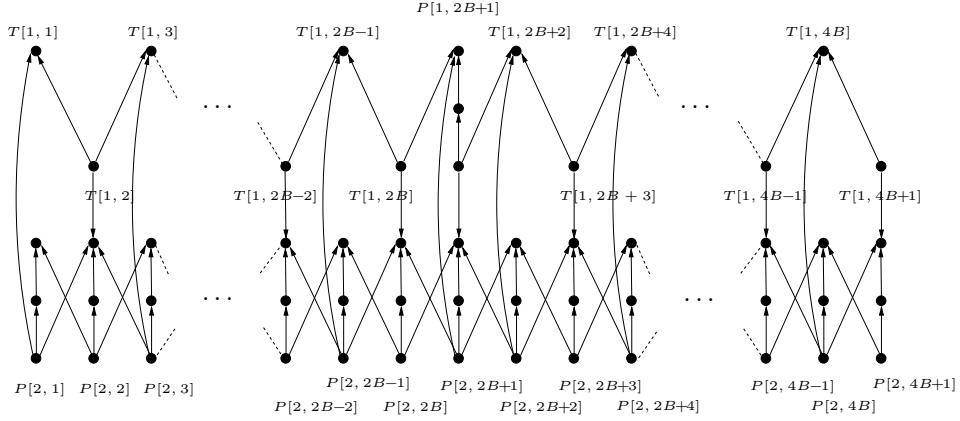


FIG. 3.2 – Illustration du squelette pour une grille à 2 lignes

Le squelette est constitué de $3q(p - 1) + (4B + 3)$ tâches.

- Chaque élément a_i de l'ensemble \mathcal{A} sera représenté par un sous-graphe du graphe de précedence G . Nous procédons de la façon suivante, l'élément a_i sera représenté par le sous-graphe X_{a_i} . Le sous-graphe X_{a_i} sera une copie du graphe $X_{v(a_i)}$ définie récursivement de la façon suivante : $\forall i > 2 : X_i = X_{i-1} @ X$ où X_2 et X sont définis de la façon suivante :

- le graphe X est composé de cinq sommets ($X[k]$ où $0 \leq k \leq 4$) et quatre arcs : $X[0] \rightarrow X[j]$ où $j \in \{2, 3, 4\}$, $X[1] \rightarrow X[4]$,
- le graphe X_2 est composé de huit sommets ($X_2[k]$ où $0 \leq k \leq 7$) et sept arcs : $X_2[0] \rightarrow X_2[4]$, $X_2[j] \rightarrow X_2[5]$ où $j \in \{0, 1, 2\}$, $X_2[2] \rightarrow X_2[l]$ où $l \in \{6, 7\}$, $X_2[3] \rightarrow X_2[7]$.

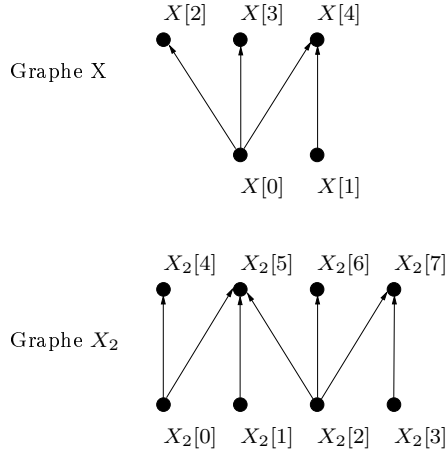


FIG. 3.3 – Illustration des graphes X et X_2

Le graphe X_j a donc par construction $4j$ sommets. L'opérateur $@$ est définie de la façon suivante :

- L'ensemble $V[X_j]$ des sommets de X_j : $V[X_j] = V[X_{j-1}] \cup V[X] \setminus$

- L'ensemble $E[X_j]$ des arcs de X_j : $E[X_j] = E[X_{j-1}] \cup E[X] \setminus \{X[0] \rightarrow X[2]\} \cup \{X[0] \rightarrow X_{j-1}[4(j-1) - 1]\}$.

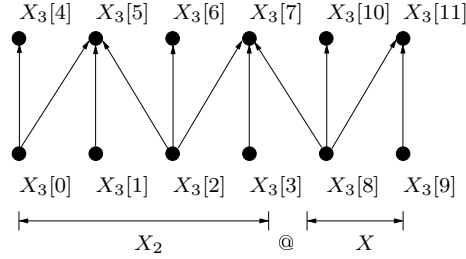


FIG. 3.4 – Illustration du graphe X_3 construit à partir de X et X_2

La transformation se fait en temps polynomial.

- Montrons tout d'abord que l'existence d'une partition de l'ensemble \mathcal{A} entraîne l'existence d'un ordonnancement valide en trois unités de temps. L'ordonnancement est construit de la façon suivante :

- Les tâches des chemins de longueur deux $P[i, j]$ du squelette s'exécutent sur le même processeur $\pi_{i,j}$.
- Les tâches $T[1, j]$ pour $j \in \{1, \dots, 4B + 1\} \setminus \{2B + 1\}$ sont exécutées sur le processeur $\pi_{1,j}$ en respectant les délais de communication, c'est-à-dire soit à $t = 2$ si $T_{i,j}$ est un puits ou soit à $t = 0$ si $T_{i,j}$ est une source.
- Les tâches des graphes associés aux éléments de $\mathcal{S} \subset \mathcal{A}$ s'exécutent sur les processeurs $\pi_{1,1}, \dots, \pi_{1,2B}$ au plus tôt.
- Les tâches des graphes associés aux éléments de $\mathcal{A} \setminus \mathcal{S}$ s'exécutent sur les processeurs $\pi_{1,2B+2}, \dots, \pi_{1,4B+1}$ au plus tôt.

- Réciproquement, montrons que l'existence d'ordonnancement de longueur trois implique l'existence d'une solution au problème *Partition*.

Lemme 3.6 *Dans un ordonnancement valide en 3 unités de temps, il n'y a pas d'instant d'inactivité.*

Preuve : Nous avons $p(4B + 1)$ processeurs et le nombre de tâches à ordonnancer est de $3p(4B + 1)$ ($(4B + 3) + 3(p - 1)(4B + 1)$ tâches du squelette et $8B$ des éléments a_i). \square

Lemme 3.7 *L'ordonnancement des tâches du squelette sature les processeurs de toutes les lignes sauf ceux de la première ligne ou de la dernière ligne.*

Preuve : Les tâches d'un chemin de longueur deux doivent être ordonnancé sur un même processeur. Lorsque deux chemins de longueur deux sont reliés par une contrainte de précédence allant d'une source à un puits, alors ces deux chemins doivent s'exécuter sur deux processeurs voisins. Supposons le contraire, le coût de communication serait supérieur à un donc nous ne pourrions pas avoir

un ordonnancement valide en trois unités de temps. Par effet de bord, il n'y a que deux possibilités pour ordonnancer les tâches du squelette. Comme ces deux possibilités sont symétriques nous allons sans perte de généralité n'en considérer qu'une :

- La tâche $T_{1,j}$ avec $j \in \{1, \dots, 4B+1\} \setminus \{2B+1\}$ s'exécute soit à $t = 0$ si c'est une source, soit à $t = 2$ si c'est un puits.
- Les tâches d'un chemin $P_{i,j} \forall i, j$ de longueur deux s'exécutent sur le processeur $\pi_{i,j}$.

□

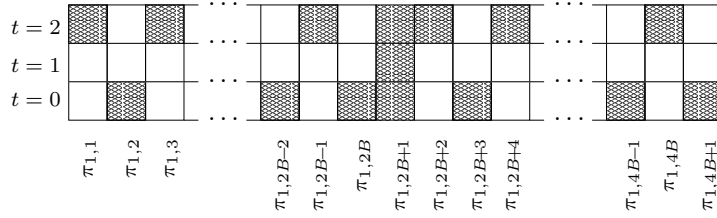


FIG. 3.5 – Processeurs partiellement libres après le placement des tâches du squelette. Un slot est grisé s'il est occupé par une tâche du squelette.

Lemme 3.8 *Le sous-graphe de précedence X_{a_j} associé à l'élément a_j a besoin de $2v(a_j)$ processeurs adjacents pour s'exécuter.*

Preuve : Par construction le sous-graphe X_{a_j} est composé de $4v(a_j)$ tâches. Après le placement du squelette nous avons vu que chaque processeur non saturé a seulement deux slots libres. En plaçant les sommets de degré entrant égaux à trois à $t = 2$, les sommets de degré sortant égaux à trois à $t = 0$ et les sommets de degré un à $t = 1$, nous avons besoin de $2v(a_j)$ processeurs pour ordonnancer X_{a_j} .

Admettons que ces $2v(a_j)$ processeurs ne soient pas consécutifs, alors l'un des coûts de communication entre une source et un puits de X_{a_j} sera strictement supérieur à un. Nous ne pourrions donc pas avoir un ordonnancement valide en trois unités de temps si le graphe X_{a_j} est ordonné sur $2v(a_j)$ processeurs non consécutifs. □

Le Lemme 3.7 nous montre que nous avons deux séries de $2B$ processeurs non saturés. Comme il y a exactement le même nombre de slots libres que de tâches à ordonnancer (voir le Lemme 3.6), pour avoir un ordonnancement valide en trois unités de temps il faut trouver un sous-ensemble de graphes X_{a_j} qui a besoin de $2B$ processeurs pour s'exécuter. D'après le Lemme 3.8, ce sous-ensemble de sous-graphe X_{a_j} représente un sous-ensemble d'éléments a_j dont la somme est égale à B .

En conclusion, si nous avons un algorithme polynomial pour notre problème d'ordonnancement, nous pourrions résoudre le problème *Partition* en temps polynomial ce qui est impossible sous l'hypothèse $P \neq NP$. □

Théorème 3.9 *Le problème de décider si une instance du problème $\{P, grille\}$ | $prec, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j)$, $dup \mid C_{max}$ possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

Preuve : La preuve vient du Théorème 3.5 et du Lemme 3.6. □

Corollaire 3.10 *Le problème $\{P, grille\} | prec, c_{ij} = d_{G^*}(\pi^i, \pi^j), p_i = 1 | C_{max}$ ne possède pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à $4/3$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve : La preuve est une conséquence immédiate du théorème de l'impossibilité ([6] et [8]). □

Remarque : Si nous prenons $p = 1$ comme donnée, nous sommes dans le cas où le graphe des processeurs est une chaîne. Nous retrouvons avec *Partition* au lieu de $m - Partition$ les résultats de [7].

De plus nous remarquons que le squelette et les graphes associés aux éléments X_{a_j} sont des arbres, par conséquent nous pouvons ajouter à ces résultats qu'avoir un arbre comme graphe de précedence ne change pas la difficulté du problème.

Corollaire 3.11 *Le problème de décider si une instance du problème $\{P, grille\}$ | $biparti, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j) \mid C_{max}$ possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

Preuve : Les sous-graphes X_{a_j} étant déjà bipartis, nous avons seulement à retoucher le squelette pour le transformer en graphe biparti. Les parties à retoucher du squelette sont les chemins de longueur deux. Nous aurons toujours trois tâches mais les contraintes de précedence vont changer. Nous avons maintenant $P[i, j][1] \rightarrow P[i, j][2]$ et $P[i, j][1] \rightarrow P[i, j][3]$ (une exemple est donné par la Figure 3.6). À présent il existe une bipartition entre les tâches sources et les tâches puits. Ces changements n'affectent pas les lemmes 3.6 et 3.8. Il est facile de voir que le lemme 3.7 est toujours valide donc avoir pour donnée un graphe de précedence biparti est aussi difficile qu'un graphe quelconque.

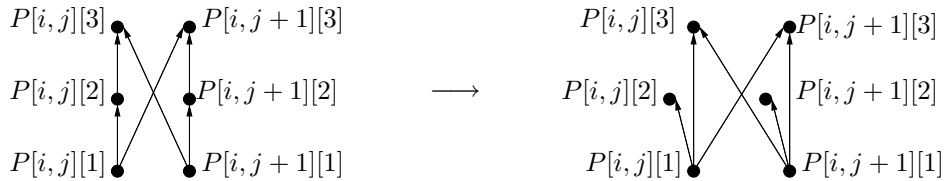


FIG. 3.6 – Illustration d'une transformation de deux chemins de longueur trois en graphe biparti

□

3.3.2 Le tore

Théorème 3.12 *Le problème de décider si une instance du problème $\{P, \text{tore}\} \mid \text{biparti}, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j) \mid C_{max}$ possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

Preuve : La preuve est identique à la preuve du théorème 3.5. La seule différence à apporter est sur le squelette. En effet, nous devons rajouter une $(4B+2)$ -ième colonne à la grille sinon nous aurions une série de $4B$ processeurs adjacents partiellement libres. Cette nouvelle colonne de processeurs, qui sera saturé par des tâches, permet de délimiter les deux séries de $2B$ processeurs nécessaire pour la partition. Enfin nous ajoutons les contraintes des précédences suivantes :

- $\forall j \in \{1, \dots, 4B+1\} \setminus \{2B+1\}$ nous avons $P[p, j][1] \rightarrow T[1, j]$ si $T[1, j]$ est un puits, $T[1, j] \rightarrow P[2, j][3]$ sinon
- $\forall j \in \{2, \dots, p\}$ nous avons $P[4B+2, j][1] \rightarrow P[1, j][3]$ et $P[1, j][1] \rightarrow P[4B+2, j][3]$

Les lemmes de la preuve du théorème 3.5 restent tous valides. Le problème est donc aussi difficile lorsque le graphe des processeurs est une grille ou un tore. \square

La transformation du squelette pour avoir un graphe biparti faites lorsque le graphe des processeurs est une grille peut se faire de la même façon pour le tore. À partir des Théorèmes 3.5, 3.12, 3.9 et du corollaire 3.11 nous pouvons regrouper ces résultats pour en tirer le corollaire suivant :

Corollaire 3.13 *Le problème de décider si une instance du problème $\{P, \alpha'\} \mid \beta, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j), \gamma \mid C_{max}$ où $\alpha' \in \{\text{grille}, \text{tore}\}$, $\beta \in \{\text{prec}, \text{biparti}\}$ et $\gamma \in \{., \text{dup}\}$ possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

3.4 La distance entre deux processeurs n'est plus unitaire

Théorème 3.14 *Le problème de décider si une instance du problème $\{P, \alpha\} \mid \beta, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j), \gamma \mid C_{max}$ où $\alpha \in \{\text{grille}, \text{tore}\}$, $\beta \in \{\text{prec}, \text{biparti}\}$, $\gamma \in \{., \text{dup}\}$ et $d_{G^*}(\pi^i, \pi^j) = \alpha \forall (i, j) \in E^*$ possède un ordonnancement de longueur $(\alpha + 2)$ est \mathcal{NP} -Complet.*

Preuve : Les résultats précédents sont facile à adapter.

Les sous-graphes X_{a_j} sont transformés pour être composés de $2v(a_j)$ chemins de longueur $\alpha + 1$ avec les contraintes de précédence illustrées par la Figure.

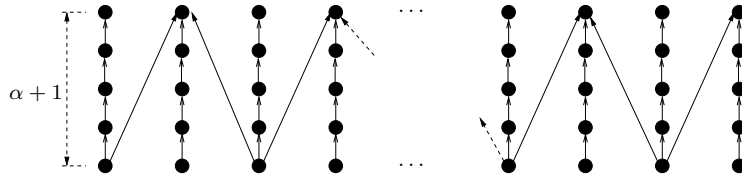


FIG. 3.7 – Illustration d'un sous-graphe X_{a_j} pour notre nouveau problème

Dans le cas où le graphe de précedence est quelconque, les chemins de longueur 2 du squelette se transforment en chemin de longueur $(\alpha + 2)$. Les tâches d'un chemin sont donc forcées d'être exécutées consécutivement.

Dans le cas où le graphe de précedence est un graphe biparti, nous procédons aux mêmes changements que dans la preuve du Théorème 3.5.

L'ensemble de ces modifications ne viole pas les lemmes 3.6, 3.7 et 3.8 donc chercher un ordonnancement valide en $\alpha + 2$ unités de temps est \mathcal{NP} -Complet dans ces conditions. \square

3.5 Conclusion

Dans ce chapitre nous avons plusieurs seuil d'inaproximabilité lorsque que le graphe des processeurs est un tore ou une grille et lorsque le graphe de précedence est quelconque, biparti ou un arbre. Nous avons redémontrer d'une autre manière les résultats de [7].

Poursuivre les recherches sur un tel modèle nous amènerai à trouver des algorithmes à garanties de performances. De même que le chapitre précédent, nous pourrions aussi généraliser le modèle en considérant que les tâches ne sont plus unitaire ou que chaque processeur ne sont pas tous aussi puissant.

Chapitre 4

L'arbre binaire équilibré

4.1 Introduction

Dans ce chapitre nous allons montrer la difficulté lorsque le graphe des processeurs est un arbre binaire équilibré. Ce chapitre a pour but de montrer que dès lors que le graphe des processeurs aura un diamètre qui dépend du nombre de sommet alors nous pourrons procéder de la même façon que la preuve de ce chapitre ou du chapitre précédent pour montrer la difficulté du problème.

4.2 Présentation de l'arbre binaire équilibré

Définition 4.1 *Un arbre binaire équilibré complet (noté ABR) de hauteur h est un arbre binaire tel que la distance de la racine à n'importe quelle feuille est égale à h .*

Propriétés :

- un ABR de hauteur h a $2^{h+1} - 1$ sommets,
- un ABR de hauteur h a 2^h feuilles.

4.3 Ordonnancement sur un arbre binaire équilibré à m processeurs

Nous allons dans un premier temps nous intéresser aux problèmes lorsque la distance entre deux processeurs voisins est unitaire. Nous ajusterons ensuite les résultats lorsque cette distance passe à $\alpha \in \mathbb{N}$.

Théorème 4.2 *Le problème de décider si une instance du problème $\{P, \text{arbre binaire}\} \mid prec, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j) \mid C_{max}$ possède un ordonnancement de longueur 2 est polynomial.*

Preuve : Il est facile de voir si un graphe de précedence peut être ordonnancé deux unités de temps. En effet, le graphe de précedence ne peut être composé que d'une collection de tâches isolées ou de chaines de longueur un. \square

Théorème 4.3 *Le problème de décider si une instance du problème $\{P, \text{arbre binaire}\} \mid prec, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j) \mid C_{max}$ possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

Preuve : Notre preuve est basée sur une réduction à partir du problème de *Partition*. La réduction est très proche de celle effectuée dans le chapitre précédent. Le seul point qui change est sur la construction du squelette. Nous détaillerons donc surtout ce point là.

Notre graphe de processeurs sera un arbre binaire de hauteur $2B$ (soit de $2^{2B+1} - 1$ sommets). Comme dans le chapitre précédent, nous allons isoler deux séries de $2B$ processeurs et bloquer les autres. L'une des deux séries de $2B$ processeurs sera dans le sous-arbre gauche issu de la racine de l'arbre tandis que l'autre série de $2B$ processeurs sera dans le sous-arbre droit.

Nous pouvons construire le squelette de façon suivante :

- Construction d'un arbre binaire équilibré de hauteur $2B$, notons ce sous-graphe $S = (V, E)$.
- Remplaçons chaque sommet de S par un chemin de longueur deux : $\forall s \in S$ on a $P[s][i]$ où $i \in \{1, 2, 3\}$ et $P[s][j] \rightarrow P[s][j+1]$ où $j \in \{1, 2\}$. Notons ce nouveau sous-graphe S' .
- Nous relient les chemins de longueur deux de la façon suivante : $\forall e = (i, j) \in E$ on crée deux arcs $P[i][1] \rightarrow P[j][3]$ et $P[j][1] \rightarrow P[i][3]$.

À ce stade nous avons un graphe de précedence qui sature tous les processeurs. Il nous reste donc à libérer deux séries disjointes de $2B$ processeurs. Nous allons les placer l'une dans le sous-arbre gauche issu de la racine, et l'autre dans le sous-arbre droit. Nous prendrons comme série de $2B$ processeurs une chaîne de processeurs allant d'une feuille du sous-arbre gauche (resp. droit) au fils gauche (resp. droit) de la racine de l'arbre.

Sur chacune des chaînes que nous avons choisi, nous partons d'un processeur à l'extrémité de la chaîne et nous enlevons successivement les tâches à $t = 0$ et $t = 1$ puis à $t = 1$ et $t = 2$ jusqu'à atteindre l'autre extrémité de la chaîne.

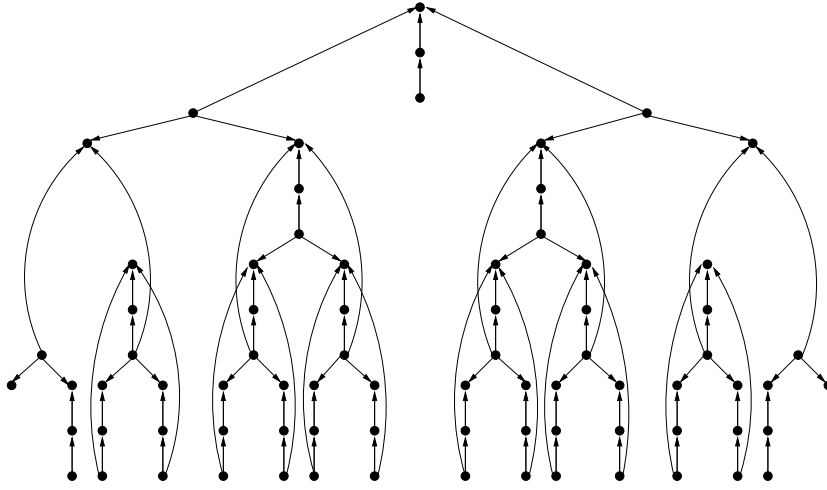


FIG. 4.1 – Illustration d'un squelette pour un arbre de hauteur 3

La construction des sous-graphes X_{a_j} reste identique que dans le chapitre précédent.

Lemme 4.4 *Dans un ordonnancement valide en trois unités de temps, il n'y a pas d'instant d'inactivité.*

Preuve : Nous avons $2^{2B+1} - 1$ processeurs soit $3 \cdot (2^{2B+1} - 1)$ slots libres et nous avons $3 \cdot (2^{2B+1} - 1) - 8B$ tâches du squelette et $8B$ tâches issues des éléments X_{a_j} . \square

Lemme 4.5 *Le placement des tâches du squelette laisse 2 séries de $2B$ processeurs disjointes partiellement libres.*

Preuve : Lorsque nous ordonnancions les tâches du squelette, les seuls choix que nous avons sont des permutations entre sous-arbres gauches et sous-arbre droit à partir d'un sommet. Cela n'affecte en rien que chacune des deux séries de $2B$ processeurs partiellement libres. \square

Le dernier lemme concernant le placement des sous-graphes X_{a_j} est toujours valide. La fin de preuve est donc en tout point identique à la preuve du chapitre précédent. \square

Théorème 4.6 *Le problème de décider si une instance du problème $\{P, \text{arbre binaire}\} | \text{prec}, p_i = 1, c_{ij} = d_G(\pi^i, \pi^j), \text{dup} | C_{max}$ possède un ordonnancement de longueur 3 est \mathcal{NP} -Complet.*

Preuve : La preuve vient du Théorème 4.3 et du Lemme 4.4. \square

Corollaire 4.7 *Le problème $\{P, \text{arbre binaire}\} | \text{prec}, c_{ij} = d_{G^*}(\pi^i, \pi^j), p_i = 1 | C_{max}$ ne possède pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à $4/3$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve : La preuve est une conséquence immédiate du Théorème de l'impossibilité (voir [6] et [8]). \square

Remarque : Nous pourrions faire les adaptations pour un graphe de précedence biparti ou lorsque de la distance entre deux processeurs voisins n'est plus unitaire de la même façon que nous avons fait pour la grille ou le tore dans le chapitre précédent.

4.4 Conclusion

Nous venons de voir une autre preuve de \mathcal{NP} – Complétude lorsque le graphe des processeurs est un arbre binaire équilibré. Cependant nous remarquons qu'elle est très similaire à la preuve du chapitre précédent. La seule différence est sur la construction du squelette, la partie importante de la preuve restant inchangée. Par conséquent si nous changeons la topologie du graphe des processeurs, pour montrer la difficulté du problème dans ces conditions (durée des tâches unitaires) il suffira de construire un nouveau squelette propre au graphe des processeurs qui sépare deux séries de processeurs pour ordonnancer les graphes associés aux éléments du problème de *Partition*.

Chapitre 5

Conclusion

5.1 Conclusion

Dans ce stage, nous avons étudié un modèle d'ordonnement où le placement des tâches a un rôle important. Nous avons étudié quelques topologies pour le graphe des processeurs.

Nous avons tout d'abord étudié le cas où le graphe des processeurs est une étoile. Nous avons montré la difficulté du problème d'optimisation associé. Nous avons proposé un algorithme approché de rapport 4 pour l'étoile et tout graphe de diamètre deux pour la minimisation de la longueur de l'ordonnement.

Nous avons ensuite montré la difficulté du problème lorsque le graphe des processeurs est une grille ou un arbre binaire. De ces deux chapitres, nous avons vu une similitude de la preuve. Ceci nous amène à penser que lorsque le diamètre du graphe des processeurs dépend du nombre de processeurs de celui-ci, nous pourrions effectuer une preuve très semblable à celle de la grille ou de l'arbre binaire équilibré.

L'ensemble des résultats de ce modèle d'ordonnement peuvent être regroupés dans le tableau ci-dessous :

G^*		G	Complexité		Borne d'approx.	
Topologie	$d(\pi^i, \pi^j)$		Poly.	$\mathcal{NP} - \mathcal{C}$	Inf.	Sup.
Chaîne	1	prec, biparti	2	3	4/3	
	α	prec, biparti	$\alpha + 1$	$\alpha + 2$	$\alpha + 3/\alpha + 2$	
Cycle	1	prec, biparti	2	3	4/3	
	α	prec, biparti	$\alpha + 1$	$\alpha + 2$	$\frac{\alpha+3}{\alpha+2}$	
Étoile	1	prec, biparti	4	5	6/5	
	α	prec, biparti	$2\alpha + 2$	$2\alpha + 3$	$\frac{2\alpha+4}{2\alpha+3}$	$\frac{4}{3} \times \left(\frac{\alpha+1}{2}\right)^2 + 1$
Grille	1	prec, biparti	2	3	4/3	
	α	prec, biparti	$\alpha + 1$	$\alpha + 2$	$\alpha + 3/\alpha + 2$	
Tore	1	prec, biparti	2	3	4/3	
	α	prec, biparti	$\alpha + 1$	$\alpha + 2$	$\alpha + 3/\alpha + 2$	
Chaîne	1	arbre	2	3	4/3	
ABR	1	prec, biparti	2	3	4/3	
	α	prec, biparti	$\alpha + 1$	$\alpha + 2$	$\alpha + 3/\alpha + 2$	

5.2 Perspectives

Une poursuite de ce stage nous amènerait à chercher des approximations lorsque le graphe des processeurs est une chaîne et décliner ces résultats pour la grille ou le tore.

Nous pourrions aussi changer légèrement le modèle en prenant en compte la puissance de chaque processeurs pour nous placer dans un cas plus proche de la réalité. Enfin nous pourrions penser que chaque sommet du graphe des processeurs n'est pas seulement un machine mais un autre groupe de travail. Ce modèle serait le regroupement entre notre modèle et le modèle à délais de communications hiérarchiques.

Bibliographie

- [1] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discr. App. Math.*, 18 :55–71, 1987.
- [2] 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.
- [3] C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comp.*, 19(2) :322–328, April 1990.
- [4] R.L. Graham, J.K. Lenstra E.L. Lawler, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory. *Ann. Discrete Math*, 1979.
- [5] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 1996.
- [6] M.R. Garey and D.S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. Freeman, 1979.
- [7] V. Boudet, Y. Cohen, R. Giroudeau, and J.C. König. Complexity results for scheduling problem with non trivial topology of processors.
- [8] P. Chrétienne and C.Picouleau. *Scheduling with Communication Delays : A Survey*, chapter 4. John Wiley & Sons, 1995.
- [9] A. Munier and J.C. König. A heuristic for a scheduling problem with communication delays. In *Operations Research*. 1997.
- [10] R. Giroudeau, J.C. König, and J. Palaysi. Complexity and approximation for the precedence constrained scheduling problem with large communications delays. LNCS, No. 3648, pages 252–261. Springer-Verlag, 2005.
- [11] Groupes thématiques de PRS Capa-Rumeur-Exec. *ICARE'97, Conception et mise en œuvre d'applications parallèles irrégulières de grande taille*, chapter 5, page 117. CNRS, décembre 1997.