Cours d'introduction à l'ordonnancement

MASTERM2 recherche Année 2006

Version 2.2

Université de Montpellier
Place Eugène Bataillon
34095 Montpellier Cedex 5

RODOLPHE GIROUDEAU
161, RUE ADA
34392 MONTPELLIER CEDEX 5
TEL : 04-67-41-85-40
MAIL : RGIROU@LIRMM.FR

Avertissement

Ce cours est une d'introduction à la théorie de l'ordonnancement. L'ordonnancement est un domaine très vaste et varié. Il existe même un journal dédié spécifiquement aux problèmes d'ordonnancement : "Journal of scheduling".

Ce cours a pour but simplement de vous familiariser avec les notations, et de vous donner quelques résultats obtenus dans le domaine de l'ordonnancement. La communauté des ordonnanceurs reste très active et régulièrement de nouveaux modèles et résultats sont présenter dans les conférences et journaux.

Table des matières

I Notations et problèmes sans communication **1**

1 Introduction et notations **3**
1.1 Introduction . . . 3
1.2 Notation des problèmes et définitions. . . . 6
1.2.1 Notation des problèmes. . . . 6
1.2.2 Définitions. . . . 7
1.2.3 $\mathcal{NP}$-complétude aux sens fort et faible . . . 8
1.2.4 Approximation à rapport dépendant de l'instance . . . 9
1.2.5 Notion de granularité . . . 10
1.2.6 Une liaison entre deux fonctions d'objectifs : la minimisation de la longueur de l'ordonnancement et la somme des temps de complétude . . . 11
1.2.7 Méthodologie . . . 13

2 Problème sans communication **19**
2.1 Introduction . . . 19
2.2 Le problème sans contrainte de précédence . . . 20
2.2.1 Complexité du problème . . . 20
2.2.2 Algorithmes d'approximation . . . 21
2.2.3 L'algorithme de Graham . . . 21
2.2.4 L'algorithme de Coffman-Graham . . . 23
2.3 Un schéma d'approximation polynomiale pour le problème $P||C_{max}$ 24
2.3.1 Définitions et rappels . . . 24
2.3.2 L'algorithme . . . 25
2.3.3 Complexité . . . 25
2.4 Programmation Dynamique . . . 27
2.4.1 Résolution de $P2||C_{max}$ par la programmation dynamique 27
2.4.2 Programmation dynamique pour le problème $P||C_{max}$ . . 28
2.5 Le problème avec des contraintes de précédence . . . 29

2.5.1 Dans le cas où le graphe est quelconque . . . 29
2.5.2 Dans le cas où le graphe est spécifié . . . 32
2.5.3 Algorithme d'approximation . . . 35
2.6 Équilibrage de charges . . . 37
2.6.1 La borne est atteinte . . . 38

II Introduction des communications **39**

3 Le modèle UET-UCT **41**
3.1 Introduction . . . 42
3.2 Problème UET-UCT avec une infinité de processeurs . . . 42
3.2.1 Complexité du problème avec graphe de précédence quelconque . . . 43
3.2.2 Seuil d'approximation pour un graphe quelconque et pour la somme des temps de complétude . . . 46
3.3 Approximation avec garantie de performance non triviale . . . 48
3.3.1 Le programme linéaire en nombres entiers . . . 49
3.3.2 Ordonnancement réalisable . . . 52
3.3.3 Analyse de la performance relative de l'algorithme . . . 54
3.3.4 La borne supérieure de la performance relative . . . 54
3.4 Problèmes avec m processeurs . . . 58
3.4.1 Complexité du problème avec un graphe de précédence quelconque . . . 58
3.4.2 Seuil d'approximation pour le problème de la minimisation de la longueur de l'ordonnancement pour un graphe quelconque . . . 61
3.4.3 Seuil d'approximation pour un graphe quelconque et pour la somme des temps de complétude . . . 65
3.4.4 Complexité dans le cas où le graphe de précédence est un arbre . . . 67
3.5 Approximation pour le problème avec m processeurs . . . 70
3.5.1 Un premier algorithme . . . 70
3.5.2 La borne supérieure est atteinte . . . 73
3.5.3 Approximation : la notion de pliage en ordonnancement . 73
3.5.4 Le pliage évolué . . . 76

4 Le modèle SCT **83**
4.1 Introduction . . . 83
4.1.1 Les petits délais de communication . . . 83

4.1.2 Introduction de la duplication 84
4.2 Le problème avec duplication 85
4.2.1 L'algorithme de Colin-Chrétienne 85
4.2.2 Justification et complexité de l'algorithme 86
4.3 Le problème sans duplication 89
4.3.1 Les petits délais de communications 89
4.3.2 Les petits délais de communication locaux 91
4.4 Conclusion 93

5 Le modèle LCT **95**
5.1 Introduction 95
5.2 Le problème avec duplication 96
5.2.1 Complexité 96
5.2.2 Approximation 99
5.3 Le problème sans duplication 101
5.3.1 Le problème avec m processeurs 101
5.3.2 Approximation 106
5.3.3 Le problème avec une infinité de processeurs 106
5.4 Non-approximability results 107
5.4.1 The minimization of length of the schedule 108
5.4.2 The special case $c = 2$ 112
5.5 A polynomial time for $C_{max} = c + 2$ with $c \in \{2, 3\}$ 114
5.5.1 Approximation 115
5.5.2 Description of the method 115
5.5.3 Analysis of the method 116
5.6 Analysis of our results 117

6 Conclusion **119**

Table des figures

1.1 *Méthodologie employée* 13
1.2 *Complexité des problèmes d'ordonnancements avec délais de communications avec pour critère la minimisation de la longueur de l'ordonnancement.* 15
1.3 *Résultats de (non−)approximation pour les problèmes d'ordonnancements avec délais de communications, avec pour critère la minimisation de la longueur de l'ordonnancement.* 17

2.1 *Illustration de la construction* $Partition \propto P_2||C_{max}$. 21
2.2 *Illustration de la preuve du Théorème 2.3.1* 26
2.3 *Exemple d'application pour le schéma d'approximation polynomiale* 27
2.4 *Exemple de transformation polynomiale* $clique \propto P|prec; p_i = 1|C_{max}$. 30
2.5 *Le graphe de précédence et l'ordonnancement associé correspond à la transformation polynomiale* $BBIS \propto P|biparti; p_i = 1|C_{max}$. 34
2.6 *Ordonnancement de liste et ordonnancement optimal* 37

3.1 *Exemple d'ordonnancement sur un système multiprocesseurs* . . . 42
3.2 *Les variables-tâches et les clauses-tâches* 44
3.3 *Construction de la transformation polynomiale pour la somme des temps de complétude à partir de celle effectuée pour la longueur de l'ordonnancement.* 47
3.4 *Exemple d'ordonnancement avec performance relative de* $\frac{4}{3}$. . . . 51
3.5 *Le graphe* B_1 *et son ordonnancement associé* $\sigma(B_1)$. 56
3.6 *Le graphe* B_2 *et son ordonnancement associé* $\sigma(B_2)$. 56
3.7 *Les tâches* V*, et l'ordonnancement optimal* 60
3.8 *Exemple de transformation polynomiale* $clique \propto P|prec; c_{ij} = 1; p_i = 1|C_{max}$. 63
3.9 *Exemple de construction pour illustrer la preuve du théorème 3.4.3* 64

3.10 *Construction de la transformation polynomiale pour la somme des temps de complétude à partir de celle effectuée pour la longueur de l'ordonnancement.* 66
3.11 *Variables-tâches et clauses-tâches correspondant à une instance du problème Satisfaisabilité avec* $c_1 = (x \vee \bar{y})$ *et* $c_2 = (\bar{x}, y)$ . . . 68
3.12 *Ordonnancement de longueur* b 70
3.13 *Un graphe de couche de type* $(5, 2)$ 71
3.14 *Graphe donnant la borne supérieure* 73
3.15 *Illustration de la preuve du théorème 3.5.2* 75
3.16 *Illustration de la preuve 1* 77
3.17 *Illustration de la preuve 2* 78
3.18 *Le graphe* $G^m(n)n > 0$ 80
3.19 *Les deux ordonnancements associés au graphe* $G^m(n)n > 0$ . . . 81

4.1 *Exemple de l'influence de la duplication sur la longueur de l'ordonnancement* 84
4.2 *Problème* P_0. 85
4.3 *Les bornes inférieures pour* P_0. 86
4.4 *Le sous-graphe critique de* P_0. 87
4.5 *L'ordonnancement au plus tôt de* P_0. 89

5.1 *Exemple de transformation polynômiale.* 97
5.2 *A partial precedence graph* 108
5.3 *Illustration of notion of an expansion* 116

Liste des Algorithmes

2.1 Un schéma d'approximation polynomiale pour le problème $P||C_{max}$ 25
3.1 *Algorithme d'arrondis* 52
3.2 *Construction de l'ordonnancement* 53
3.3 Ordonnancement sur m à partir d'un ordonnancement sur une infinité de processeurs : première méthode 74
3.4 Algorithme qui donne les date d'exécution des tâches 76
4.1 Algorithme qui détermine les bornes inférieures d'exécutions, RT 85
4.2 *Construction de l'ordonnancement optimal (Earliest Schedule)* . . 88
4.3 Algorithme donnant les dates de début d'exécution 91
5.1 Algorithme donnant les bornes inférieures $e(v)$ d'exécution d'une tâche v 99
5.2 Algorithme donnant un orodnnancement réalisable 99
5.3 Algorithme donnant la solution optimale 101

Première partie

Notations et problèmes sans communication

CHAPITRE

1 Introduction et notations

Sommaire

1.1 Introduction 3
1.2 Notation des problèmes et définitions. 6
1.2.1 Notation des problèmes. 6
1.2.2 Définitions. 7
1.2.3 $\mathcal{NP}$-complétude aux sens fort et faible 8
1.2.4 Approximation à rapport dépendant de l'instance . . . 9
1.2.5 Notion de granularité 10
1.2.6 Une liaison entre deux fonctions d'objectifs : la minimisation de la longueur de l'ordonnancement et la somme des temps de complétude 11
1.2.7 Méthodologie 13

1.1 Introduction

Depuis quelques années, l'informatique est présente et rythme notre quotidien. En effet, il suffit de constater le taux d'équipement des ménages en ordinateur, l'évolution du monde automobile, tant au niveau de la conception par l'intermédiaire des outils de C.A.O., que de la fabrication avec la robotisation, de l'aide à la conduite avec les ordinateurs de bord, du pilotage automatique des avions, de la connaissance en temps réel des bouchons, etc.

Si le $XX^{ième}$ siècle était celui des machines et de la Taylorisation, le siècle qui s'ouvre à nous sera celui de la communication, des échanges de toutes sortes via les réseaux à haut-débits. Mais avec cette expansion, de nouveaux problèmes apparaissent liés à la discipline elle-même (où le parallélisme, l'internet et la globalisation de l'environnement de l'information en sont la face apparente), et également

avec l'apparition de problèmes calculatoires importants issus d'autres disciplines (analyse de séquences en biologie, simulations numériques en physique, etc).

L'utilisation du parallélisme (c'est-à-dire l'emploi de plusieurs processeurs) pour le traitement des applications de grande taille qui réclament une puissance de calcul de plus en plus importante est aujourd'hui une réalité. En effet, il n'est plus à démontrer l'intérêt du parallélisme pour le traitement des grandes applications issues de le physique (simulations en physique nucléaire) ou le traitement des séquences de nucléotides en biologie. Des applications de ce type ne peuvent pas être traitées en un temps raisonnable sur une machine séquentielle. Dans le but de les traiter le plus rapidement possible, les solutions techniques qui ont été développées pour le parallélisme sont des architectures parallèles avec divers choix architecturaux : des architectures parallèles avec mémoire partagée (les processeurs se partagent une mémoire centrale), à mémoire distribuée (chaque processeur dispose d'une mémoire), des machines vectorielles, etc. Ces dernières années, il existe un regain d'intérêt pour le parallélisme avec l'apparition et l'utilisation de plus en plus croissante des grappes de stations de travail comme machine parallèle.

Néanmoins, les puissances de calcul théoriques des machines parallèles ne sont, en pratique, jamais atteintes. Ceci est dû principalement aux difficultés liées à la gestion des ressources et des contraintes de fonctionnement des architectures multiprocesseurs. Parmi les difficultés que l'on rencontre, on peut citer l'étape d'extraction du parallélisme intrinsèque d'une application ou les problèmes de routage et d'ordonnancement des communications entre les différents processeurs de l'architecture.

Pour tenter de pallier à la première difficulté, une phase de partitionnement est nécessaire. Elle correspond à la première étape fondamentale de la parallélisation et consiste en l'extraction du parallélisme potentiel d'une application.

L'extraction du parallélisme détermine les dépendances fonctionnelles entre les différentes parties d'une application. Ainsi, une application parallèle peut être représentée sous forme d'un graphe orienté sans circuit (graphe de précédence) où chaque sommet du graphe représente une partie (instruction ou ensemble d'instructions) du programme et les arcs, les contraintes chronologiques entre ces parties (dites aussi contraintes de précédence). 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.

Formellement, si nous appelons t_i la date de début d'exécution de la tâche i et si nous notons par π_i le processeur sur lequel s'exécute la tâche i, si (i, j) est un arc du graphe de précédence :

- $t_j - t_i \geq p_i$ si i et j sont exécutées sur le même processeur, c'est-à-dire si $\pi_i = \pi_j$,

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.

Plusieurs modèles prennent en compte les délais de communications entre les différentes parties d'une application dont le plus populaire est celui qui est connu comme le modèle à *communications homogènes*. Dans ce modèle nous supposons que tous les processeurs de la machine parallèle sont totalement connectés : si deux tâches communicantes i et j s'exécutent sur deux processeurs différents alors la communication potentielle c_{ij}, entre ces deux tâches, est indépendante des processeurs sur lesquels elles s'exécutent. La localisation de l'exécution des tâches i et j n'influe pas sur la communication c_{ij}. Il est important de noter que la prise en compte de la topologie de la machine parallèle induit un degré de difficulté supplémentaire rendant les problèmes d'optimisation très difficiles car le placement des tâches sur les processeurs devient une composante fondamentale dans le but d'obtenir un ordonnancement réalisable et de "bonne qualité".

Dans le modèle avec *communications homogènes*, nous pouvons distinguer deux types de communications :

- les *communications intra-processeurs* : ce sont les communications entre deux tâches adjacentes dans le graphe de précédence, qui s'exécutent sur le même processeur (en respectant les contraintes de précédence). Nous supposerons ces délais négligeables.
- les *communications inter-processeurs* : ce sont les communications entre deux tâches adjacentes dans le graphe de précédence qui s'exécutent sur des processeurs différents.

Formellement, nous considérons un graphe orienté sans cycle $G = (V, E)$ avec V l'ensemble des sommets et E l'ensemble des arcs. Chaque tâche $i \in V$ a une durée d'exécution de p_i et à chaque arc (i, j) est associée une valeur représentant le délai de communication potentiel c_{ij} entre les tâches i et j. Soit t_i (resp. t_j) la date de début d'exécution de la tâche i (resp. j), alors si i et j sont exécutées sur le même processeur alors nous avons $t_j \geq t_i + p_i$, sinon si elles sont exécutées sur des processeurs différents, alors $t_j \geq t_i + p_i + c_{ij}$.

Ce modèle a été très largement étudié ces dernières années (recherche de solutions approchées, résultats de complexité, algorithmes optimaux ...), comme en témoignent les nombreux articles publiés sur le domaine (voir le chapitre suivant pour la présentation des principaux résultats sur le *modèle homogène*, le livre [8] et l'article de synthèse [6]).

1.2 Notation des problèmes et définitions.

1.2.1 Notation des problèmes.

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. [12] et par Błażewicz et al. [4]. Nous allons étendre les notations à notre modèle.

Pour définir un problème d'ordonnancement, nous avons besoin d'introduire 3 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 α définit la présence ou non de processeur, le type de processeur et leur nombre.
  $*\alpha \in \{P, \bar{P}, P_m\}$
  - Si $\alpha = P$ signifie qu'on a des processeurs identiques et que leur nombre m est une entrée du problème.
  - Si $\alpha = \bar{P}$ ou $\alpha_2 = \infty$ signifie qu'on a des processeurs identiques, leur nombre étant suffisant ou non bornée.
  - Si $\alpha = P_m$ signifie qu'on a un nombre de processeurs identiques fixé.
- Le paramètre β définit le type de 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 péremption.
  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 β_1 =prec (le graphe est quelconque).
  - Si β_1 =arbre (le graphe est un arbre)
  - $\vdots$
  - Si $\beta_1 = .$ (les tâches sont indépendantes)

  $*\beta_2 \in \{com, c_{jk}, c, .\}$
  - Si β_2 =com (les temps de communication sont données 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 tâche).
  - $\beta_2 = .$ (il n'existe pas de communication).

  $*\beta_3 \in \{p_j, .\}$
  - 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 β_4 =dup (on autorise la duplication des tâches).
- Si $\beta_4 = .$ (on n'autorise pas la duplication des tâches).

$*\beta_5 \in \{pmtp, .\}$
- Si β_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 le temps d'achèvement maximum noté C_{max} comme étant le $max(t'_1, \ldots, t'_n)$ où n est le nombre de tâches avec $t'_j = t_j + p_j$ où p_j représente la durée d'exécution de la tâche j et t_j représente le début de la date d'exécution de j.

1.2.2 Définitions.

La plupart des problèmes d'ordonnancement avec temps de communications sont des problèmes difficiles. Une possibilité pour les résoudre consiste à développer des heuristiques qui calculent des ordonnancements réalisables dont la durée est proche de celle d'un ordonnancement optimal.

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

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

Définition 1.2.1 (Ratio de performance ou compétivité.) *Soit h un algorithme d'ordonnancement. Le ratio de performance de h, noté $R(h)$, est le maximum sur tous les graphes entre la durée donnée par un ordonnancement fourni par h et la durée de l'ordonnancement optimal, c'est-à-dire*

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

On dit qu'un algorithme h est $\rho-$approché (ou encore $\rho-$compétitif) si $\rho(h) \leq \rho$.

La valeur du ratio de performance permet de cerner la complexité pratique du problème et de traduire l'augmentation de la complexité du problème en fonction

des nouveaux paramètres intégrés au modèle. D'un point vue pratique, l'heuristique de meilleur ratio n'est pas forcément celle à intégrer si le surcoût induit par la mise en œuvre s'avère trop important.

Un algorithme h est d'autant plus efficace que son ratio $\rho(h)$ est petit. La démarche dans le cas d'un problème $\mathcal{NP}$-complet est donc de trouver la meilleure heuristique possible. En fait la plupart du temps nous ne sommes capables que de donner une borne supérieure du ratio de la meilleur heuristique (en étudiant le ratio d'une heuristique connue) et une borne inférieure de ce ratio.

Une technique classique pour obtenir une borne inférieure du ratio est basée sur le résultat suivant, facile à démontrer et appelé "théorème de l'impossibilité" [8] :

Théorème 1.2.1 *Étant donné un problème d'ordonnancement et un entier c, si la question de savoir si un graphe G peut-être ordonnancé en c unités de temps ou moins est $\mathcal{NP}$-complet alors on ne peut pas espérer avoir une heuristique pour ce problème d'ordonnancement ayant un ratio inférieur à $\frac{c+1}{c}$.*

Preuve

Si nous supposons qu'il existe un tel algorithme A, alors celui-ci peut-être utilisé pour décider si $OPT(I) \leq c$ en un temps polynomial. Nous supposons que l'algorithme A est un algorithme ρ approché pour le problème d'ordonnancement avec $\rho < \frac{(c+1)}{c}$
- Si $OPT(I) \leq c$, alors $A(I) < \frac{(c+1)}{c} OPT(I) \leq c + 1$. Sachant que l'algorithme détermine une solution réalisable ayant pour valeur comme fonction d'objectif une valeur entière et nous obtenons $A(I) \leq c$.
- Si $OPT(I) \geq c + 1$, alors $A(I)$ est également au moins de $c + 1$.

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

□

1.2.3 $\mathcal{NP}$-complétude aux sens fort et faible

Soit Π un problème et I une instance de taille $|I|$. Notons par $\max(|I|)$ le plus grand nombre qui apparaît dans I. Remarquons que $\max(|i|)$ peut être exponentiel en $|I|$. Ceci ne pose évidemment pas un problème de représentation polynomiale de ce nombre puisque, le codage binaire (tel est le codage de chaque nombre dans un ordinateur) d'un nombre x nécessite $O(\lceil \log x \rceil)$ bits.

Définition 1.2.2 *Un algorithme pour Π est appelé pseudo-polynomial s'il est polynomial en $|I|$ et $\max(|I|)$ (si $\max(|I|)$ est exponentiel $|I|$, alors cet algorithme peut être exponentiel pour I).*

Définition 1.2.3 *Un problème d'optimisation est $\mathcal{NP}$-complet au sens fort (strongly $\mathcal{NP}$-complete) si le problème est $\mathcal{NP}$-complet à cause de sa structure et non pas à cause de la 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 et polynomialement borné. Un problème est $\mathcal{NP}$-complet au ses faible (weakly $\mathcal{NP}$-complete) s'il est $\mathcal{NP}$-complet à cause de ses variables (c'est à dire,* $\max(|I|)$ *intervient dans la complexité des algorithmes qui le résolvent).*

Théorème 1.2.2 *Si un problème* Π *est $\mathcal{NP}$-complet au sens fort, alors il n'est pas résoluble par un algorithme pseudo-polynomial, sauf si $\mathcal{NP} = \mathcal{P}$.*

Preuve

Supposons qu'il existe un algorithme optimal de complexité pseudo-polynomial. Cet algorithme serait alors de complexité polynomial en $\max(|I|)$, et cette quantité étant toujours polynomial en $|I|$, par conséquent, cet algorithme serait un algorithme polynomial optimal pour ce problème, ceci est impossible si $\mathcal{NP} \neq \mathcal{P}$ □

1.2.4 Approximation à rapport dépendant de l'instance

Les rapports dépendant de l'instance sont souvent des fonctions soit de la taille même de l'instance d'un problème, soit d'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.
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 si il existe un algorithme polynomial approché A pour un problème Π et une constante fixée $r \in \mathbb{R}^+$ tels que le rapport d'approximation classique de A est borné par r.
4. **PTAS** Un schéma polynomial d'approximation (polynomial time approximation schema) 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 complètement polynomial d'approximation (fully polynomial time approximation schema) 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 si il admet un schéma d'approximation classique complètement polynomial.

Nous démontrons maintenant un résultat qui relie la $\mathcal{NP}$-complétude au sens fort (et la pseudo-polynomialité) avec la possibilité d'existence d'un schéma complet d'approximation polynomiale.

Théorème 1.2.3 *Soit* Π *un problème d'optimisation à valeurs entières $\mathcal{NP}$-difficile au sens fort. S'il existe un polynôme p tel que, pour toute instance I de* Π, $opt(I) < p(|I|, \max(I))$ *alors* Π *n'a pas de schéma d'approximation complètement polynomial sauf si $\mathcal{P} = \mathcal{NP}$.*

Preuve

Soit Π un problème de maximisation (le cas de la minimisation se traître de manière identique). Supposons que pi admet un schéma d'approximation complètement polynomial A_ϵ (c'est à dire, $\forall \epsilon > 0$, l'algorithme A_ϵ est polynomial en $|I|$ et en $1/\epsilon$). Posons $B = A_\epsilon$ pour $\epsilon = 1/p(|I|, \max(I))$ fixé. Alors B est polynomial en $|I|$ et $p(|I|, \max(I))$ c'est à dire que, B est pseudo-polynomial. Soit S_B une solution déterminée par B sur I. Nous avons donc $m(I, S_B) \geq opt(I)(1-\epsilon)$ (puisque A_ϵ est un schéma d'approximation complètement polynomial) et $opt(I) - m(I, S_B) \leq opt(I) - (1-\epsilon)opt(I) \leq \epsilon opt(I) < \frac{1}{p(|I|, \max(I))} p(|I|, \max(I)) = 1$.

Puisque les valeurs réalisables de Π sont entières, $opt(I) - m(I, S_B) < 1$ implique que $opt(I) - m(I, S_B) = 0$. Ceci est absurde étant donné que dans ce cas B est exact et Π est $\mathcal{NP}$-difficile au sens fort.

□

1.2.5 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 taille des calculs effectués par une tâche et les délais de communications.

Définition 1.2.4 *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 :*

$$\alpha = \frac{\max c_{ij}, (i,j) \in E}{\min p_i, i \in V}$$

Dans le cas où $\alpha \geq 1$ (resp. $\alpha < 1$) alors nous dirons que le problème d'ordonnancement est 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.

En effet, si nous considérons une tâche $i \in V$ alors $\forall y \in \Gamma^+(i), \forall k \in \Gamma^+(i)$ (où $\Gamma^+(i)$ désigne l'ensemble des successeurs de i), nous avons $p_j \geq c_{ik}$ alors, dans tout ordonnancement réalisable, la tâche i aura au plus un successeur j tel que le temps de communication entre i et j soit supprimé. Autrement dit, il y aura au plus un successeur j de i tel que $t_i + p_i + c_{ij} > t_j$.

De manière symétrique, si pour une tâche $i \in V$ alors $\forall y \in \Gamma^-(i), \forall k \in \Gamma^-(i)$ (où $\Gamma^-(i)$ désigne l'ensemble des prédécesseurs de i), nous avons $p_j \geq c_{ik}$ alors, dans tout ordonnancement réalisable, la tâche i aura au plus un prédécesseur j tel que le temps de communication entre i et j soit supprimé. Autrement dit, il y aura au plus un prédécesseur j de i tel que $t_j + p_j + c_{ji} > t_i$.

Une définition locale de la granularité plus générale a été introduite par Gérasoulis et Yang [3].

La plupart des résultats théoriques ont été obtenus (voir les figures 1.2 e 1.3) dans les cas des petits délais de communications. La définition donnée ci-dessus englobe le cas où les tâches et les communications ont des durées unitaires (i.e. $\forall i \in V,\ p_i = 1$ et $\forall (i,j) \in E,\ c_{ij} = 1$).

1.2.6 Une liaison entre deux fonctions d'objectifs : la minimisation de la longueur de l'ordonnancement et la somme des temps de complétude

La technique du "gap" consiste à proposer une réduction créant un "gap" sur la valeur de la fonction objectif des instances construites.

Nous utilisons un résultat classique de la théorie de la complexité qui consiste à proposer une $Turing$-réduction d'un problème de décision (dans notre cas le problème d'ordonnancement avec pour minimisation la longueur de l'ordonnancement,) à un problème d'optimisation (ici le problème d'ordonnancement avec pour critère la minimisation de la somme des temps de complétude) qui partionnent les valeurs du problème d'optimisation en deux sous-ensembles clairement séparés, l'un correspondant à des instances, du problème de décision, pour lesquels la réponses est oui et l'autre à celles pour lesquelles la réponse est non. De plus, le premier de ces ensembles contient les « grandes » valeurs et le deuxième les « petites », de façon que si M_0 est la plus petite des « grandes » valeurs et m_0 la plus grande des « petites » valeurs, alors nous avons l'inégalité stricte $M_0 > m_0$; en

d'autres termes il existe un saut discret ente les valeurs m_0 et M_0. Cette technique de séparation claire de valeurs, qui ne marche pas pour toutes réduction, s'appelle « technique de saut » (gap technique).

Cette technique est la plus utilisé pour l'obtention de résultats d'inapproximabilité.

On considère un problème Π de décision et un problème Π' d'optimisation (supposons qu'il soit de minimisation) ; nous supposons qu'un algorithme A garantit un rapport d'approximation ρ pour le second. Alors, pour toute instance I de Π, la réduction construit une instance I' de Π' pour laquelle il existe un nombre c tel que :

- si I est une instance positive, alors $opt(I') \leq c$, ce qui implique $A(i') \leq \rho c$;
- si I est une instance négative, alors $opt(I') > \rho c$, ce qui implique $A(I') \geq opt(I') > \rho c$.

Nous disons alors que cette réduction a produit un gap de valeur ρ, qui implique qu'un algorithme A garantissant un rapport d'approximation ρ pour Π' ne peut pas exister, à moins que $\mathcal{P} = \mathcal{NP}$, autrement dit Π pourrait être décidé (résolu) en temps polynomial.

Définition 1.2.5 *Soient Π et Π' deux problèmes de minimisation. Une GAP réduction de : Pi à Π avec paramètres (c, ρ) et (c', ρ'), fonctions de I et I', respectivement, et avec $\rho, \rho' \geq 1$, est une réduction $R = (f, g)$ (la fonction f transformant une instance I de Π en une instance $I' = f(I)$ et la fonction g transformant une solution réalisable $S' \in SOL_{\Pi'}(f(I))$ en une solution $S = g(I, S') \in SOL_{\Pi}(I)$, les deux fonctions f et g étaant polynomiale en $\max\{|I|, |I'|\}$ telle que les valeurs des solutions optimales sur I et I' satisfont la propriété suivante :*

- *si $opt(I) \geq c(I)$, alors $opt(I') \geq c'(I')$;*
- *si $opt(I) < c(I)\rho(I)$, alors $opt(I') < c'(I')\rho'(I')$*

Comment une telle réduction peut être utilisée pour produire des résultats négatifs pour Π' ? Supposons que l'on ait conçu par ailleurs une réduction $R(f', g')$ (polynomiale) entre un problème de décision Π_0 et Π qui affirme que, pour toute instance I_0 de Π_0 :

- si I_0 admet une réponse oui alors $opt(f'(I_0)) \geq c(I_0)$;
- sinon alors $opt(f'(I_0)) < c(I_0)\rho(I_0)$

Cette réduction implique bien évidemment, l'inapproximabilité de Π avec un rapport ρ à moins que $\mathcal{P} = \mathcal{NP}$. Si on construit la composition $GAP \circ R$ où GAP-réduction est telle qu'elle a été définie dans la définition 1.2.5 avec paramètres (c, ρ) et (c', ρ'), alors nous obtenons une réductio de Π_0 à Π pour laquelle la propriété suivante est vérifiée : pour toute instance I_0

– si I_0 admet une réponse oui alors $opt(f(f'(I_0))) \geq c'(I_0)$ (bien évidemment, $f(f'(I_0))$ est une instance de I' de Π') ;
– sinon alors $opt(f(f'(I_0))) < c'(I_0)\rho'(I_0)$

Alors, avec le même raisonnement que précédemment, nous pouvons conclure immédiatement qu'il n'existe pas d'algorithme polynomial garantissant un rapport ρ' pour Π' sauf si $\mathcal{P} = \mathcal{NP}$.

Une technique classique pour obtenir une borne inférieure du facteur d'approximation, pour un problème donné, est basée sur la technique du "gap" ([13]).

Cette technique sera utilisée dans les sections 3.2.2 et 3.4.3 lorsque nous voudrons étendre les résultats de non-approximabilité au cas où le critère est la minimisation de la somme des temps de complétude. Nous utiliserons la solution obtenue quand le critère de minimisation est la longueur de l'ordonnancement (C_{max}) pour l'étendre au cas de la minimisation de la somme des temps de complétude ($\sum_j C_j$).

Pour obtenir les résultats de non-approximabilité qui sont présentés dans cette thèse, nous utilisons la technique du "gap".

1.2.7 Méthodologie

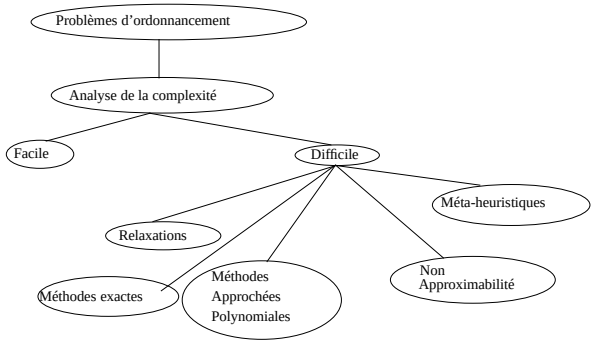


FIG. 1.1 – *Méthodologie employée*

Lorsqu'on aborde un problème d'ordonnancement, et de manière plus générale un problème en recherche opérationnelle, dans un premier temps on s'intéresse à la complexité de ce problème :

– Dans le cas où le problème s'avère facile, le principe est de développer un algorithme ayant la plus faible complexité qui résout celui-ci.
– dans le cas où le problème s'avère difficile ($\mathcal{NP}$-complet, $\mathcal{NP}$-dur, $\mathcal{APX}$, ...), plusieurs orientations s'offrent à nous :
  – On peut relaxer les contraintes du problème, (par exemple passage d'un nombre fini de processeurs, à un nombre infini de processeurs), et on analyse maintenant le problème relaxé.
  – On peut déterminer des résultats de non-approximabilité c'est-à-dire dans une borne k pour laquelle, pour n'importe quel algorithme A, ayant une performance relative ρ^A, on a $\rho^A > k$. Nous obtenons ainsi un seuil d'approximation, sous lequel nous ne pouvons pas trouver d'algorithme.
  – Il n'est pas seulement intéressant de connaître la complexité d'un problème, il est souhaitable de développer des algorithmes polynomiaux pour le résoudre de manière approchée.
  – On peut également développer des méta-heuristiques, pour tenter de résoudre au mieux un problème d'ordonnancement : recherche locale, méthode tabu, programmation dynamique, ...

Dans ce cours, nous nous sommes focalisé sur les résultats de non-approximabilité et sur la recherche d'algorithme polynomiaux approchés ayant une performance relative non triviale. Les figures 1.2 et 1.3 donnent les principaux résultats des problèmes d'ordonnancement avec communications dans le cadre du `modèle homogène`. La figure 1.2 donne la complexité des problèmes d'ordonnancements avec délais de communications avec pour critère la minimisation de la longueur de l'ordonnancement. Nous pouvons remarquer que la majeure partie des problèmes sont $\mathcal{NP}$-complet dans le cas général. Le seul qui échappe à cette règle c'est le cas du problème d'ordonnancer sur une infinité de processeurs, en autorisant la duplication, un graphe de précédence quelconque soumis à l'hypothèse des petits délais de communications. Ce problème sera étudié dans la section 4.1 de ce cours. Il est important de noter que lorsque le graphe de précédence est spécifier (arbres, biparti, ...), il existe des solutions polynomiales.

La figure 1.3 présent les résultats de (non−)approximation pour les problèmes d'ordonnancements avec délais de communications, avec pour critère la minimisation de la longueur de l'ordonnancement. On peut noter qu'un grand nombre d'algorithmes polynomiaux donnant une solution approchée ont été proposés. Il reste es problèmes ouverts : nous pouvons remarquer que le problème de déterminer la complexité du problème $Pm|prec; c_{ij} = 1; p_i = 1|C_{max}$ avec un nombre de machines fixé est ouvert même dans le cas où $m = 2$ et la complexité de $\bar{P}|prec; c_{ij} = c > 1; p_i = 1|C_{max}$ n'a pas été encore déterminé.

FIG. 1.2 – *Complexité des problèmes d'ordonnancements avec délais de communications avec pour critère la minimisation de la longueur de l'ordonnancement.*

FIG. 1.3 – *Résultats de (non−)approximation pour les problèmes d'ordonnancements avec délais de communications, avec pour critère la minimisation de la longueur de l'ordonnancement.*

CHAPITRE

2 Problème sans communication

Sommaire

2.1 Introduction 19
2.2 Le problème sans contrainte de précédence 20
2.2.1 Complexité du problème 20
2.2.2 Algorithmes d'approximation 21
2.2.3 L'algorithme de Graham 21
2.2.4 L'algorithme de Coffman-Graham 23
2.3 Un schéma d'approximation polynomiale pour le problème $P||C_{max}$ 24
2.3.1 Définitions et rappels 24
2.3.2 L'algorithme 25
2.3.3 Complexité 25
2.4 Programmation Dynamique 27
2.4.1 Résolution de $P2||C_{max}$ par la programmation dynamique 27
2.4.2 Programmation dynamique pour le problème $P||C_{max}$ 28
2.5 Le problème avec des contraintes de précédence 29
2.5.1 Dans le cas où le graphe est quelconque 29
2.5.2 Dans le cas où le graphe est spécifié 32
2.5.3 Algorithme d'approximation 35
2.6 Équilibrage de charges 37
2.6.1 La borne est atteinte 38

2.1 Introduction

La théorie de l'ordonnancement s'intéresse à l'allocation optimale des différentes parties d'une application sur les ressources machines afin que l'application

soit accomplie le plus rapidement et/ou au moindre coût. Devant la diversité des machines parallèles existantes et dans un souci d'indépendance de la machine cible, nous avons essayé de classifier et d'étudier les modèles de machines en prenant en compte les *paramètres* les plus importants des systèmes considérés (architectures parallèles, réseaux hiérarchiques, etc).

Exemple : On considère une équipe de cinq astronautes qui se préparent à faire une sortie dans l'espace. Ils doivent effectuer un certain nombres de tâches. Chaque tâche ne peut être "faite" que par un seul astronaute, et certaines tâches ne peuvent se faire qu'après certaines tâches se soient terminées. Quel tâche devra traitée par quel astronaute tel que l'ensemble des tâche soit traité le plus tôt possible ?

Dans ce chapitre, nous allons présenter un résultat de complexité et deux algorithmes d'approximations pour le problème qui consiste à ordonnancer un ensemble de tâches indépendantes ayant chacune une certaine durée, sur un nombre limité de machine noté $P||C_{max}$. Ces résultats datent d'environ 30 ans.

Rappel de la définition :

Si (i, j) est un arc du graphe de précédence :

- $t_j - t_i \geq p_i$ si i et j sont exécutées sur le même processeur,
- au plus une tâche est exécutée sur un processeur à un instant donné.

2.2 Le problème sans contrainte de précédence

2.2.1 Complexité du problème

Théorème 2.2.1 *Le problème $P||C_{max}$ est $\mathcal{NP}$-complet.*

Preuve

On va montrer que $P_2||C_{max}$ est $\mathcal{NP}$-complet. La démonstration sera basé sur le problème **Partition** :

Données : $A = \{a_1, \ldots, a_n\}$ et une valeur $s(a_i) \in \mathbb{N}$, pour chaque $a_i \in A$.

Question : Existe-il un sous-ensemble $A' \subset A$ tel que $\sum_{a_i \in A'} s(a_i) = \sum_{a_i \in A - A'} s(a_i)$?

Étant donnée une instance quelconque du problème **Partition**, on peut créer une instance de $P_2||C_{max}$ de la manière suivante :

- $n = |A|$ où n est le nombre de tâches
- $p_j = s(a_j), j = 1, \ldots, n$
- $C_{max} = \frac{1}{2} \sum_{j=1}^{n} p_j$.

Il est évident qu'il existe un sous-ensemble A' pour l'instance de la partition si et seulement si il existe un ordonnancement de longueur inférieur à $C_{max} = \frac{1}{2} \sum_{j=1}^{n} p_j$ pour $P_2||C_{max}$.

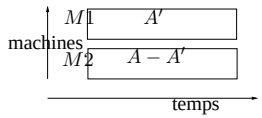


FIG. 2.1 – *Illustration de la construction* $Partition \propto P_2||C_{max}$.

□

2.2.2 Algorithmes d'approximation

Dans cette partie, nous allons proposer deux algorithmes d'approximation avec des garanties de performances non triviales. Ces deux algorithmes se basent sur le principe de maintenir sous forme d'une liste les tâches ordonnançables à un instant t. L'analyse de ces algorithmes est basée sur la comparaison entre un certaine borne inférieure de la qualité de la solution optimale, la comparaison serait meilleure sur la solution optimale mais cette valeur on ne peut la déterminer. Il faut se contenter que d'une borne inférieure.

2.2.3 L'algorithme de Graham

Rappel du problème

Un problème d'ordonnancement à m machines

Données : n tâches $p_1, p_2, \cdots, p_n$ et $<= (I, U)$ une relation résumant les contraintes de précédence entre ces tâches.

Question : Ordonnancer ces tâches sur m machines identiques en une durée totale minimale notée C_{max}.

Par la suite on va utiliser un ordonnancement par liste.

Principe :

- On construit une liste de priorité des tâches
- A chaque étape la première machine disponible est sélectionnée pour exécuter la première tâche de la liste.

Exemple : Pour l'exemple suivant on considère que les tâches ont aucune contrainte de précédence. On suppose que $n = 13$ et que $(P_1, P_2, \cdots, P_{12}, P_{13}) = (1, 1, \cdots, 1, 4)$. Proposer un ordonnancement dans le cas où la liste des priorités est la suivante $(T_1, T_2, \cdots, T_{12}, T_{13})$ sur quatre machines. Donner l'ordonnancement optimale.

Théorème 2.2.2 *Pour n'importe quel ordonnancement de liste* LS *on a* $\frac{C_{max}^{LS}}{C_{max}^{*}} \leq 2$.

Preuve

Nous allons nous basé sur les deux bornes inférieures suivantes :

$$C_{max}^{*} \geq \sum_{j=1}^{n} p_j/m = \frac{T_{seq}}{m} \tag{2.1}$$

$$C_{max}^{*} \geq p_j \text{ pour toutes les tâches} \tag{2.2}$$

Soit j' la tâche qui se termine à C_{max}^{LS} et soit $t_{j'}$ le début d' exécution de la tâche j'. Ainsi, nous obtenons $C_{max}^{LS} = t_{j'} + p_{j'}$. Toutes les machines sont au travail avant $t_{j'}$ car sinon on aurait pu commencer l'exécution de la tâche j' plus tôt. Le temps maximum d'utilisation des machines est $\sum_{j=1}^{n} p_j/m$, donc

$$C_{max}^{LS} \leq t_{j'} + p_{j'} \tag{2.3}$$

$$\leq \sum_{j=1}^{n} p_j/m + p_{j'} \tag{2.4}$$

$$\leq C_{max}^{*} + C_{max}^{*} = 2C_{max}^{*} \tag{2.5}$$

□

2.2.3.1 La borne supérieure est atteinte

Nous pouvons construire une instance pour laquelle la borne supérieure est atteinte :

Nous considérons $n(n-1)$ tâches de durées d'exécution unitaire et une tâche de durée n. Nous supposons que nous disposons de $m = n$ machines. Les deux ordonnancements extrêmes sont les suivants :

- L'ordonnancement optimal consiste a exécuter la plus longue tâche sur une machine et l'autre sur les $(n-1)$ restantes entre les instants $[0, n]$. Toutes les machines sont actives et le taux d'occupation est de 100%, donc la longueur de l'ordonnancement optimal est $C_{max}^{*} = n$.
- Le mauvais ordonnancement consisterait à exécuter les $n(n-1)$ tâches unitaires sur tous les machines et d'exécutée en dernier la plus longue tâche, ainsi la longueur de l'ordonnancement sera de $C_{max}^{LS} = n - 1 + n$.

2.2.4 L'algorithme de Coffman-Graham

Principe :
- On construit une liste de priorité des tâches dans l'ordre décroissant.
- à chaque étape la première machine disponible est sélectionnée pour exécuter la première tâche de la liste.

Théorème 2.2.3 *Pour n'importe quel ordonnancement de liste on a* $\frac{C_{max}^{LPT}}{C_{max}^{*}} \leq \frac{4}{3} - \frac{1}{3m}$.

Preuve

Nous allons commencer par faire une simplification du problème. Supposons que j' est la dernière tâche qui finit son exécution dans l'ordonnancement, mais j' n'est pas la dernière tâche à commercer son exécution, c'est à dire $\exists j_1, \ldots, j_k$ tel que $t_{j_i} > t_{j'}$, $\forall i \in \{1, \ldots, k\}$. Nous pouvons supprimer toutes ces tâches sans affecter la longueur de l'ordonnancement sachant que ces tâches sont exécutées sur d'autres machines. Donc le fait d'enlever ces tâches ne peut que faire décroître que la longueur de l'ordonnancement optimal. Ainsi, si nous prouvons la borne d'approximation pour cette nouvelle instance,cette borne reste valide pour l'instance initiale.

Sans perte de généralité on suppose que $p_1 \geq p_2 \geq \ldots \geq p_n$. Sans perte de généralité, on peut considérer que l est la dernière tâche de la liste de priorité. Si ceci n'est pas le cas, on peut tout simplement oublier les tâches qui suivent et ainsi obtenir une nouvelle instance pour laquelle LPT se comporte de la même manière par rapport à l'ordonnancement optimal. Soit l la tâche qui se termine en dernier dans l'ordonnancement LPT.

- **Premier cas :** $p_l \leq \frac{C_{max}^{*}}{3}$. On a

$$C_{max}^{LPT} \leq \frac{1}{m}\sum_j p_j + \frac{m}{m-1}p_l \leq C_{max}^{*} + \frac{m}{m-1}\frac{C_{max}^{*}}{3} = (\frac{4}{3} - \frac{1}{3m})C_{max}^{*}$$

  La première partie de l'inégalité est vraie car nous sommes toujours dans le cas d'un ordonnancement par liste. De plus nous savons que $T_{iddle} = \frac{m}{m-1}p_l$ et comme précédemment $C_{max}^{LPT} = \frac{T_{seq}+T_{iddle}}{m}$.
- **Second cas :**$p_l > \frac{C_{max}^{*}}{3}$.
  Pour une telle instance, dans un ordonnancement optimal de longueur C_{max}^{*}, au plus deux tâches peuvent s'exécuter sur n'importe quel processeur. ($p_l > \frac{C_{max}^{*}}{3}$ et toutes les autres tâches ont des durées $\geq p_l$).
  - Si $n \leq m$ alors trivialement nous exécutons une seule tâche par machine.

  - Nous supposons maintenant que le nombre de tâches $m < n \leq 2m$. Supposons donc que l'instance comporte $2m - P_h$ tâches où $0 \leq P_h < m$ d'où $n = 2m - P_h$. Il est facile de comprendre que l'ordonnancement suivant est optimal :
    - les tâches $1, \ldots, P_h$ sont exécutés seuls chacune sur un processeur différents.
    - chaque couple de tâches $(h+1, n), (h+2, n-1), \ldots$ est exécuté sur un processeur différent (il reste $n - P_h = 2m - P_h \rightarrow n = 2(m - P_h)$ tâches a exécutées pour $(m - P_h)$ machines de libre.

    Or cet ordonnancement est l'ordonnancement fournit par LPT. Dans ce cas $p_l > \frac{C_{max}^{*}}{3}$ LPT est optimal.

□

2.2.4.1 La borne est atteinte

Dans cette partie nous allons montrer que la borne est atteinte pour un certain type de graphe, ce qui implique que les calculs proposés précédemment sont les plus justes possibles.

Exemple : La tâche m précède les tâches $m+1,\ m+2, \ldots,\ 2m$. La tâche $2m+1$ suit les tâches $m+1,\ m+2, \ldots,\ 2m$. Les durées des tâches sont les suivantes :

- $p_1 = p_2 = \ldots = p_n = (2m-1, 2m-1, 2m-2, 2m-2, \ldots, m+1, m+1, m, m, m)$

La solution optimale est $C_{max}^{*} = m + \epsilon$ tandis que la solution donnée par l'algorithme de liste LS donne $C_{max}^{LS} = 2m - 1$. Donc en faisant tendre ϵ vers zéro on obtient bien le rapport souhaité.

2.3 Un schéma d'approximation polynomiale pour le problème $P||C_{max}$

2.3.1 Définitions et rappels

Dans cette partie, nous allons rappeler la définition d'un schéma d'approximation polynomiale.

Définition 2.3.1 *$A(\epsilon)$ est un schéma d'approximation si et seulement si pour chaque valeur de ϵ et instance I du problème, $A(\epsilon)$ génère une solution réalisable tel que : $\frac{C^*(I) - C^h(I)}{C^*(I)} \leq \epsilon$ avec $C^*(I)$ une solution optimale et $C^h(I)$ une solution donnée par notre heuristique.*

Définition 2.3.2 *Un schéma d'approximation est un schéma d'approximation polynomial si et seulement si pour chaque $\epsilon > 0$, le temps d'exécution est polynomial à la taille du problème. Par exemple $O(n^{c.\frac{1}{\epsilon}})$.*

2.3.2 L'algorithme

Algorithme 2.1 Un schéma d'approximation polynomiale pour le problème $P||C_{max}$

Soit k un entier fixé
Ordonner les tâches par ordre croissant de durée décroissante
Déterminer, par une recherche exhaustive, un ordonnancement optimale des k première tâches
Compléter l'ordonnancement par un procédé glouton en attribuant à chaque étape la tçache suivante à la machine la moins chargée

Remarque : Cet algorithme est bien évidemment polynomial tant que k est constant.

2.3.3 Complexité

Théorème 2.3.1 *L'algorithme 2.3.2 est un schéma d'approximation polynomiale pour le problème $P||C_{max}$.*

Preuve

Nous savons que $C^{opt}_{max}(I) \geq \frac{1}{m}\sum_{i=1}^{n} p_i$.

Soit K la valeur de $C^{opt}_{max}(I') = K$ où I' est l'instance du problème correspondant aux k premières tâches. Il est clair que $C^{h}_{max}(I) \geq C^{opt}_{max}(I) \geq K$.

Deux cas se présentent à nous :

– soit $C^{h}_{max}(I) = K$ alors l'instance est résolue de manière optimale par recherche exhaustive.
– ou $C^{h}_{max}(I) > K$.

Dans la suite, nous considérons le cas où $C^{h}_{max}(I) > K$.

Il existe une tâche j avec $j > k$ (les k premières tâches ont été ordonnancées de manière optimales) tel que $C^{h}_{max}(I) = t_j + p_j$. Ainsi, aucun processeur est inactif entre $[0, C^{h}_{max}(I) - p_j]$.

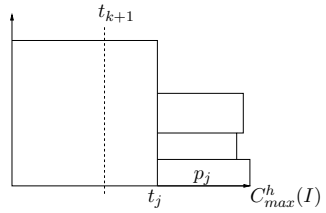


FIG. 2.2 – *Illustration de la preuve du Théorème 2.3.1*

Pour toutes les tâches j avec $j > k$ (ou $j \geq k+1$) alors $p_j \leq p_{k+1}$ car les tâches sont triées par ordre décroissant.

Soit

$$\begin{aligned}
\sum_{i=1}^{n} p_i &\geq m \times (C^{h}_{max}(I) - p_j) + p_j \\
&\geq mC^{h}_{max}(I) - (m-1) + p_j \\
&\geq mC^{h}_{max}(I) - (m-1) + p_{k+1} \\
mC^{opt}_{max}(I) &\geq mC^{h}_{max}(I) - (m-1)p_{k+1} \\
C^{h}_{max}(I) &\leq C^{opt}_{max}(I) + \frac{m-1}{m}p_{k+1}
\end{aligned}$$

Maintenant il faut trouver un majorant de p_{k+1} en rapport avec $C^{opt}_{max}(I)$. Or nous avons $\frac{k}{m}p_{k+1} \leq C^{opt}_{max}(I)$ (je prends la tâche $(k+1)$, on sait que $p_i \geq p_{k+1},\ 1 \leq i \leq k$, et donc la valeur de l'ordonnancement optimale ne peut faire mieux que l'équivalent de 100% de taux d'occupation rapporter par rapport à la tâche $(k+1)$).

Nous arrivons donc à $\frac{C^{h}_{max}(I)}{C^{opt}_{max}(I)} \leq 1 + \frac{m-1}{k}$

Il suffit de fixer $\epsilon > 0$, de choisir $k_\epsilon \geq \lceil \frac{m-1}{\epsilon} \rceil \geq \frac{m-1}{\epsilon}$ pour garantir le rapport d'approximation $1+\epsilon$.

Remarque : m est constant ceci implique que k_ϵ est constant.

Cette analyse est valide pour toute instance. Nous avons ainsi développer une famille d'algorithme, qui constitue un schéma d'approximation polynomiale pour le problème $P||C_{max}$.

□

Théorème 2.3.2 *La complexité de l'algorithme 2.3.2 est de $O(n \log n + p^{k_\epsilon})$.*

Preuve

La troisième étape de l'algorithme 2.3.2 nécessite une complexité $O(p^{k_\epsilon})$. Globalement, la complexité de l'algorithme est de $O(n \log n + p^{k_\epsilon})$. Par conséquent, sa complexité globale est polynomiale pour tout ϵ ne dépendant pas de valeurs de paramètres de l'instance. Elle devient cependant exponentielle dès que ϵ dépend de telles valeurs.

□

Exemple :

Soit $m = 2$, $n = 6$ et $k = 4$. Considérons la liste des tâches à ordonnancer $(P_1, \ldots, P_6) = (8, 6, 5, 4, 4, 1)$. L'ordonnancement pour les k premières tâches, l'ordonnancement complet et l'ordonnancement optimale est donné par la figure 2.3.

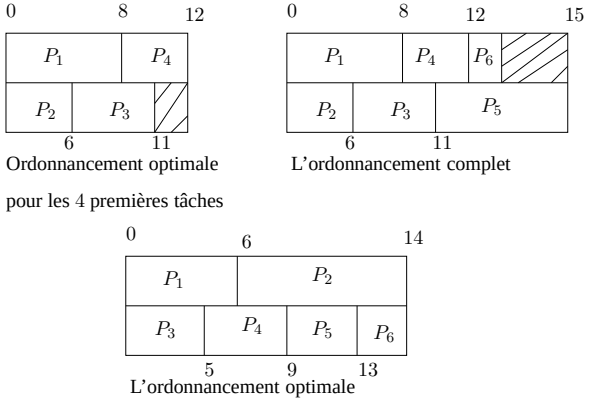


FIG. 2.3 – *Exemple d'application pour le schéma d'approximation polynomiale*

2.4 Programmation Dynamique

2.4.1 Résolution de $P2||C_{max}$ par la programmation dynamique

Exemple : Considérons le problème $P2||C_{max}$ avec cinq tâches suivant :

tâches	1	2	3	4	5
p_j	7	8	2	4	1

Nous voulons savoir s'il existe une partition des tâches correspondant à la moitié de la durée totale, c'est à dire 11 dans notre exemple. Nous allons donner un résultat optimal à l'aide de la programmation dynamique.

2.4.1.1 Algorithme

Soit F_j tel que :

$$F_j(z) = \begin{cases} 1 & \text{si } z = 0 \; \forall j \\ 1 & \text{si } z = p_1 \text{ et } j = 1 \\ 1 & \text{si } \exists \text{ x tel que } F_{j-1}(x) = 1 \text{ et } (x + p_j) = z \\ 0 & \text{sinon} \end{cases}$$

Autrement dit, F_j nous indiquera 1 s'il existe un ensemble de tâches $(1, 2, \ldots, j-1, j)$ tel que la somme des durées soient exactement z et 0 sinon. Les valeurs de $F_j(z)$ doivent être déterminée pour $j = 1, \ldots, 5$ et $z = 0, \ldots, 11$ dans notre exemple. Le programme utilise une matrice M à valeurs dans $0, 1$ et la remplie ligne par ligne.

z	0	1	2	3	4	5	6	7	8	9	10	11
j=1	1	0	0	0	0	0	0	1	0	0	0	0
j=2	1	0	0	0	0	0	0	1	1	0	0	0
j=3	1	0	1	0	0	0	0	1	1	1	1	0
j=4	1	0	1	0	1	0	1	1	1	1	1	1
j=5	1	1	1	1	1	1	1	1	1	1	1	1

2.4.1.2 Complexité

Cet algorithme s'exécute en temps polynomial et est de l'ordre de $O(n \sum \frac{p_j}{2})$. Il dépend donc de la taille des données en $O(\sum p_j)$ et est de ce fait pseudo-polynomial.

2.4.2 Programmation dynamique pour le problème $P||C_{max}$

Pour un second exemple d'utilisation de la notion de programmation dynamique, nous considérons le problème $P||C_{max}$, et nous allons nous focaliser sur un cas spécial (le nombre de tâches ayant des durées d'exécution diféérentes est bornée par une constante) admettant un algorithme polynomial.

Lemme 2.4.1 *Soit une instance du problème $P||C_{max}$ dans laquelle les durées d'exécution p_j admettent au plus s valeurs distinctes, il existe alors un algorithme qui détermine la solution optimale en $n^{O(s)}$.*

Preuve

Supposons que nous avons une longueur d'ordonnancement donné T. Nous allons utiliser la programmation dynamique.

Soient $z_1,\ z_2, \ldots,\ z_s$ les différentes durées d'exécution. Le point central est le fait que l'ensemble des tâches exécutées sur une machine peut être décrit par un vecteur $V = (v_1, \ldots, v_s)$ de dimension s, où $v_k$représente le nombre de tâches de longueur z_k.

Il existe au plus n^s vecteurs sachant que le nombre de tâches en tout est n. Soit $\mathcal{V}$ l'ensemble de tous ces vecteurs dont la somme des durées d'exécution ($\sum v_i z_i$) est inférieur ou égal à T. Dans un ordonnancement optimal, sur chaque processeur est affecté un ensemble de taches correspondant à un vecteur de $\mathcal{V}$.

Nous définnissons maintenant $M(x_1, \ldots, x_s)$ comme étant le nombre minimum de processeurs nécessaires pour ordonnancer un ensemble de tâches constitués de x_i tâches de taille z_i avec $i = 1, \ldots, s$.

Nous pouvons observer que $M(x_1, \ldots, x_s) = 1 + \min_{V \in \mathcal{V}} M(x_1 - v_1, \ldots, x_s - v_s)$.

La minimisation se fait tous les vecteurs possibles. Le premier processeur reçoit le premier vecteur, et l'appel récursif décrit la suite des opérations.

Nous avons besoin de calculer une table avec n^s entrées, où chaque entrée dépend des $O(n^s)$ autres entrées, et par la même le calcul est en $O(n^{2s})$.

Tout ceci est basé sur le fait que nous avons fixé T. IL suffit de faire varier T pour résoudre le problème C_{max}. □

2.5 Le problème avec des contraintes de précédence

2.5.1 Dans le cas où le graphe est quelconque

Dans cette partie nous allons nous intéresser au problème noté $P|prec; p_j = 1|C_{max}$. Le résultat suivant est du à Lenstra, Rinnoy Kan en 1978.

Théorème 2.5.1 *Le problème de décider si une instance de $P|prec; p_i = 1|C_{max}$ possède un ordonnancement de longueur au plus 3 est $\mathcal{NP}$-complet.*

Preuve

La démonstration sera basé sur le problème **Clique** :

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

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

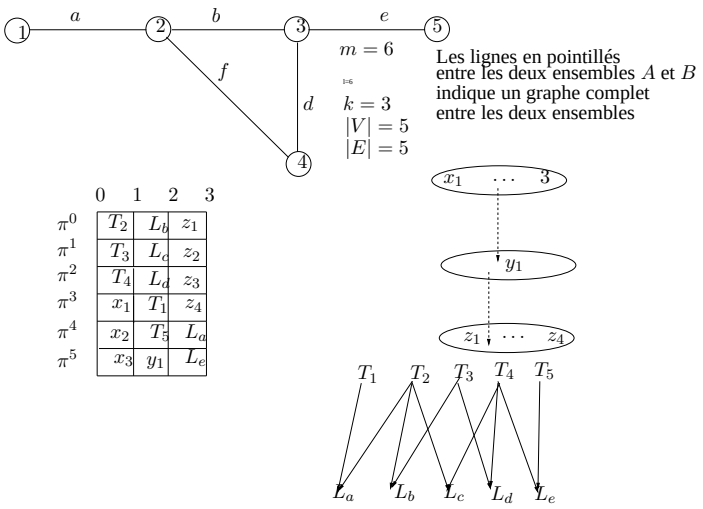


FIG. 2.4 – *Exemple de transformation polynomiale $clique \propto P|prec; p_i = 1|C_{max}$.*

Il est facile de voir que le problème $P|prec; p_i = 1|C_{max}$ est dans $\mathcal{NP}$. Notre preuve est basée sur la réduction $Clique \propto P|prec; p_i = 1|C_{max}$.

Soit une instance π^* de Clique, nous allons construire une instance π de $P|prec; p_i = 1|C_{max} = 3$ de la manière suivante :

Soit $G = (V, E)$ un graphe. Soit l le nombre d'arêtes d'une clique de taille k. Soit $k' = |V| - k$ et $l' = |E| - l$

- $\forall v \in V$ on introduit une tâche T_v et pour chaque arête $e \in E$ on introduit une tâche L_e. On a que $T_v \rightarrow L_e$ si v est une extrémité de e.
- On introduit également 3 ensembles de tâches :
  - $X_x = \{x_1, \ldots, x_{m-k}\}$
  - $Y_y = \{y_1, \ldots, y_{m-l-k'}\}$
  - $Z_z = \{z_1, \ldots, z_{m-l'}\}$

  Les contraintes de précédence entre ces tâches sont les suivantes $X_x \rightarrow Y_y \rightarrow Z_z, \forall x, yz$.
- Soit $m = \max\{k, l + k', l'\} + 1$

• Nous supposons qu'il existe une clique de taille k dans le graphe G. Montrons alors qu'il existe un ordonnancement en trois unités de temps.

Pour cela, il suffit de considérer l'ordonnancement suivant :

- à $t = 0$ nous exécutons les k tâches représentant les k sommets de la clique sur k processeurs et sur les $m - k$ processeurs nous exécutons les tâches de l'ensemble X_x.
- à $t = 1$ nous exécutons les l tâches représentant les l arêtes de la clique. Sur k' autres processeurs nous exécutons les k' dernières tâches de T_v. Enfin, sur les $m - l - k'$ derniers processeurs, nous exécutons les tâches de l'ensemble Y_y.
- à $t = 2$, nous exécutons les l' tâches représentant les arêtes ne faisant pas partie de la clique, et nous ordonnançons les $m - l'$ tâches de l'ensemble Z_z sur $m - l'$ processeurs.

Il est clair que l'affectation des tâches de la manière proposé ci-dessus donne une ordonnancement valide en trois unités de temps.

• Réciproquement, supposons qu'il existe un ordonnancement en trois unités de temps, montrons alors que le graphe G contient une clique de taille k.

- Il est clair qu'avec les contraintes entre les ensembles X_x, Y_y et Z_z, les tâches de X_x (resp. Y_y) s'exécutent forcément à $t = 0$ (resp. à $t = 1$). Et donc, les tâches de Z_z à $t = 2$. De plus, à l'instant $t = 0$ (resp. à $t = 1$) après exécutions des tâches de X_x (resp. Y_y) il ne reste que k (resp. $k' + l$) processeurs de libre. De même, à l'instant $t = 2$, il y a l' processeurs de libre.
- Il est clair également que les tâches $T_v, \forall v \in V$ ne peuvent s'exécutées soit à $t = 0$ et/ou $t = 1$.
- Les tâches de $T_e, \forall e \in E$ ne peuvent s'exécutées aux instants $t = 1$ et/ou $t = 2$.

- Nous avons $3m$ instants pour $3m$ tâches à ordonnancer, alors dans l'ordonnancement valide tous processeurs sont actifs. IL n'existe pas d'instant d'inactivité.

Soit V_1 (resp. E_1) le sous-ensemble des sommets (resp. d'arêtes) associés aux tâches qui s'exécutent à l'instant $t = 0$ (resp. $t = 1$). Toute arête de E_1, a forcément ces deux extrémités dans le sous-ensemble V_1, donc est un ensemble de l arêtes reliant k sommets, et donc les k sommets de l'ensemble V_1 forment une clique.

□

Corollaire 2.5.1 *Il n'existe pas d'algorithme d'approximation A garantissant $\frac{C^A_{max}}{C^{opt}_{max}} < \frac{4}{3}$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve

La preuve du Corollaire 2.5.1 est une conséquence immédiate du théorème de l'impossibilité (voir [10],[8]).

□

2.5.2 Dans le cas où le graphe est spécifié

Dans cette partie nous allons nous intéresser au problème noté $P|biparti; p_i = 1|C_{max}$ qui consiste à ordonnancer un graphe de précédence avec des durées unitaires sur un nombre de machines limités. Ainsi nous montrons que dans le cas où au problème $P||C_{max}$ (c'est-à-dire ordonnancer un ensemble de tâches indépendantes sur un nombre de machines limitées), nous imposons que les tâches soient soumises à des contraintes de précédence alors même dans le cas des graphes bipartis (graphe 2-coloriable) le problème est $\mathcal{NP}$-complet. Ainsi, nous montrons que même dans le cas où les tâches sont soumises à des contraintes de précédence "simples" le problème devient difficile.

2.5.2.1 Complexité

Pour déterminer la complexité du problème nous allons utiliser le problème suivant :

Le problème BBIS est un problème $\mathcal{NP}$-complet, voir ([10],[29]).

Définition 2.5.1 Instance du problème BBIS :

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 appartient à X et k à Y ?

Si un tel sous-ensemble existe, nous l'appellerons *un sous-ensemble indépendant équilibré* (BBIS) d'ordre k. BBIS reste $\mathcal{NP}$-complet même si $k = \frac{n}{2}$ (voir [29]). Par la suite nous appellerons ce problème, le problème du stable équilibré.

Théorème 2.5.2 *Le problème de décider si une instance de $P|biparti; p_i = 1|C_{max}$ possède un ordonnancement de longueur au plus 3 est $\mathcal{NP}$-complet.*

Preuve

Il est facile de voir que le problème $P|biparti; p_i = 1|C_{max}$ est dans $\mathcal{NP}$. Notre preuve est basée sur la réduction $BBIS \propto P|biparti; p_i = 1|C_{max}$.

Soit une instance π^* de BBIS, nous allons construire une instance π de $P|biparti; p_i = 1|C_{max} = 3$ de la manière suivante :

- $B = X \bigcup Y$ où $X = \{x_1,\ x_2, \ldots,\ x_n\}$ et $Y = \{y_1,\ y_2, \ldots,\ y_n\}$ et où les arêtes entre les sous-ensembles X et Y sont remplacées par les arcs correspondants orientés des sommets de X vers les sommets de Y.
- Nous ajoutons deux ensembles de tâches : $W = \{w_1,\ w_2, \ldots,\ w_{n/2}\}$ et $Z = \{z_1,\ z_2, \ldots,\ z_{n/2}\}$. Les contraintes de précédence entre ces tâches sont les suivantes :

$$w_i \rightarrow z_j,\ \forall i \in \{1,\ 2, \ldots,\ n/2\},\ \forall j \in \{1,\ 2, \ldots,\ n/2\}.$$

- Nous ajoutons également les contraintes de précédence suivantes :

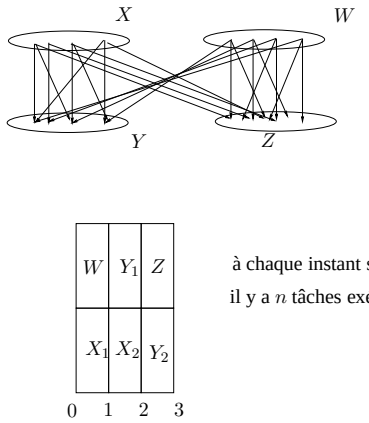
$$w_i \rightarrow y_j,\ \forall i \in \{1,\ 2, \ldots,\ n/2\},\ \forall j \in \{1,\ 2, \ldots, n\ \}.$$

et

$$x_i \rightarrow z_j,\ \forall i \in \{1,\ 2, \ldots,\ n\},\ \forall j \in \{1,\ 2, \ldots, n/2\ \}.$$

- Nous supposons que $m = n$ où m représente le nombre de processeurs.

La construction est illustrée par la figure 2.5. La transformation proposée ci-dessus est une transformation calculable en temps polynomial.



à chaque instant sur l'ensemble des processeurs il y a n tâches exécutées.

FIG. 2.5 – *Le graphe de précédence et l'ordonnancement associé correspond à la transformation polynomiale $BBIS \propto P|biparti; p_i = 1|C_{max}$.*

- Supposons que le graphe B admette un stable équilibré d'ordre k avec $n/2$ sommets noté (X_2, Y_2) où $X_2 \subset X$ et $Y_2 \subset Y$ et $|X_2| = |Y_2| = n$. Alors, il existe un ordonnancement des tâches de G en trois unités de temps. Construisons l'ordonnancement associé.
  - Nous exécutons à $t = 0$ les $n/2$ tâches de l'ensemble $X - X_2 = X_1$ et les $n/2$ tâches de l'ensemble W.
  - Nous exécutons à $t = 1$ les $n/2$ tâches de l'ensemble X_2 et les $n/2$ tâches de l'ensemble Y_2.
  - Nous exécutons à $t = 2$ les $n/2$ tâches de l'ensemble Y_1 et les $n/2$ tâches de l'ensemble Z.

  On remarque que l'ordonnancement proposé ci-dessus préserve les contraintes de précédence et les délais de communications et produit un ordonnancement de longueur trois quand il existe un stable équilibré d'ordre k dans le graphe B.
- Réciproquement, nous supposons maintenant qu'il existe un ordonnancement de longueur au plus 3. Nous allons prouver que l'existence d'un tel ordonnancement nécessite l'existence d'un stable équilibré d'ordre k, $(X'_1,\ Y'_1)$, dans le graphe B, où $X'_1 \subset X$ et $Y'_1 \subset Y$ et $|X'_1| = |Y'_1| = n/2$. Nous procédons dans un premier temps à quelques remarques essentielles.

Remarquons que dans n'importe quel ordonnancement de longueur au plus trois :

1. sachant que le nombre de tâches est exactement égal à $3n$, alors dans n'importe quel ordonnancement de longueur trois tous les processeurs sont actifs.
2. toutes les tâches de l'ensemble W doivent avoir un début d'exécution à $t = 0$. En effet, nous supposons qu'une tâche de l'ensemble W a un début d'exécution à $t = 1$, alors avec les contraintes de précédence, la longueur de l'ordonnancement n'est pas respectée.
3. à $t = 0$ aucune tâche de Y ou de Z ne peut commencer. Soit X_1' le sous-ensemble des tâches de X qui commencent leur exécution à $t = 0$. Il est clair que $|X_1'| = \frac{n}{2}$.
4. Aucune tâche de Y ne peut être exécuté à $t = 0$ et aucune tâche de X à $t = 2$.

Nous notons X_1', l'ensemble des tâches de l'ensemble X qui sont exécutées à $t = 1$ et par Y_1' l'ensemble des tâches de l'ensemble Y qui sont exécutées à $t = 1$ sur des processeurs différents. Avec les remarques précisées ci-dessus l'ensemble $X_1' \bigcup Y_1'$ doit être un stable équilibré d'ordre $\frac{n}{2}$.

Ceci achève la démonstration du théorème 2.5.2.

□

Corollaire 2.5.2 *Il n'existe pas d'algorithme d'approximation A garantissant $\frac{C_{max}^A}{C_{max}^{opt}} < \frac{4}{3}$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve

La preuve du Corollaire 2.5.2 est une conséquence immédiate du théorème de l'impossibilité (voir [10],[8]).

□

2.5.3 Algorithme d'approximation

Nous allons montrer que l'algorithme de liste proposé dans la partie 2.2.2 reste valide est conduit à un algorithme $2 - \frac{1}{m}$-approché.

Théorème 2.5.3 *Pour n'importe quel ordonnancement de liste LS on a $\frac{C_{max}^{LS}}{C_{max}^*} \leq 2 - \frac{1}{m}$.*

Preuve

On note par C_{max}^* la longueur de l'ordonnancement optimale.

- Dans un premier temps nous allons donner deux bornes inférieures pour C_{max}^* : On a trivialement que $C_{max}^* \geq \frac{T_{seq}}{m}$ (avec T_{seq} désignant le temps séquentiel) et $C_{max}^* \geq \sum_{p=1}^{k} P_{i_p}$ où P_{i_p} est un chemin de tâches définie ultérieurement.
- Nous voulons montrer que $T_{idle} \leq (m-1)\sum_{p=1}^{k} P_{i_p}$ où P_{i_p} est la durée d'exécution d'une tâche i_p faisant partie d'un chemin de tâches dans l'ordonnancement que nous allons définir. Pour cela considérons la tâche T_{i_1} dont la fin d'exécution coïncide avec C_{max}^{LS}. Pour cela considérons la tâche T_{i_1} dont la fin d'exécution coïncide avec C_{max}^{LS}. Et soit $T_{i_2} = T_j \backslash T_j \in \Gamma^-(T_{i_1}) \wedge C_j = max_{T_i \in \Gamma^-(T_{i_1})} C_i$ où $C_i = t_i + p_i$ désigne la fin d'exécution de la tâche i et $\Gamma^-(T_{i_j})$ l'ensemble des prédécesseurs de i_j. Il n' y a pas de temps d'inactivité entre la fin d'exécution de T_{i_2} et le début de T_{i_1} sinon on aurait pu avancé le début de l'exécution de T_{i_1}. En procédant ainsi de suite on a $T_{i_k} = T_j \backslash T_j \in \Gamma^-(T_{i_{k-1}}) \wedge C_j = max_{T_i \in \Gamma^-(T_{i_{k-1}})} CT_i$.
- Est-ce qu'il existe des processeurs (ou machines) inactifs entre la fin d'exécution d'un prédécesseur de T_{i_1} et le début de T_{i_1}. Entre CT_{i_k} et le début d'exécution de $T_{i_{k-1}}$ tous les processeurs sont actifs. A chaque temps t au moins un processeur est actif donc le temps d'inactivité est $(m-1)$, en faisant la somme sur les éléments du chemin on obtient ce qu'on veut.
- $C_{max}^{LS} = \frac{T_{seq}+T_{idle}}{m}$ et $T_{seq} \leq \sum_{i=1}^{n} P_i$ et donc $C_{max}^{LS} \leq \frac{\sum_{i=1}^{n} P_i}{m} + (1 - \frac{1}{m})\sum_{i=1}^{k} P_i$. Sachant que $\frac{\sum_{i=1}^{n} p_i}{m} \leq C_{max}^*$ on obtient $C_{max}^{LS} \leq C_{max}^* + (1 - \frac{1}{m})C_{max}^*$

□

2.5.3.1 La borne supérieure est atteinte

Dans cette partie nous allons montrer que la borne est atteinte pour un certain type de graphe, ce qui implique que les calculs proposés précédemment sont les plus justes possibles.

Exemple : La tâche m précède les tâches $m+1,\ m+2, \ldots,\ 2m$. La tâche $2m+1$ suit les tâches $m+1,\ m+2, \ldots,\ 2m$. Les durées des tâches sont les suivantes :

- $p_1 = p_2 = \ldots = p_{m-1} = m-1$
- $p_m = \epsilon$
- $p_{m+1} = p_{m+2} = \ldots = p_{2m} = 1$
- $p_{2m+1} = m-1$

La solution optimale est $C^*_{max} = m + \epsilon$ tandis que la solution donnée par l'algorithme de liste LS donne $C^{LS}_{max} = 2m - 1$. Donc en faisant tendre ϵ vers zéro on obtient bien le rapport souhaité.

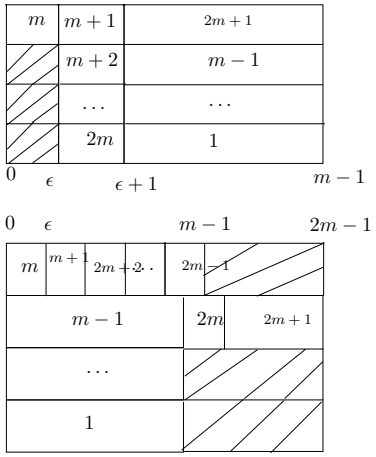


FIG. 2.6 – *Ordonnancement de liste et ordonnancement optimal*

2.6 Équilibrage de charges

Un autre problème que l'on peut rencontrer lorsque l'on étudie les problèmes d'ordonnancement c'est le problème d'équilibrage de charges c'est-à-dire que l'on dispose d'un ensemble de machines et nous souhaitons que chaque machine est à peu près la même charge. Nous allons le problème d'équilibrage de charges pour le problème avec un ensemble de tâches indépendantes. Ce problème revient en fait à un problème de Bin-packing. Rappel du problème :

Données : n articles $u_1, u_2, \cdots, u_n$ de tailles $s(u_1), s(u_2), \cdots, s(u_n)$ avec $s(u_i), (i = 1, n)$ compris entre 0 et 1.

Question : Mettre les n articles dans un nombre minimum de boites de capacité 1.

Une idée pour résoudre consiste a mettre l'article i dans la boîte admissible de plus petit indice (méthode appelé Next Fit).

Nextt Fit

{ Les articles sont indicés selon une liste quelconque }
Pour $i = 1$ à n faire
Déterminer le plus petit indice de boite j pour lequel la place restant disponible est supérieure à $s(u_i)$.
Affecter l'article u_i à la boite j.

Théorème 2.6.1 *Next Fit est un algorithme 2-approchée.*

Preuve Si on appelle $F^*(I)$ la valeur d'une solution optimale alors la borne inférieure est égale à $F^*(I) \geq \sum_{i=1}^{n} s(u_i)$. Soit $s(u_1), s(u_2), \cdots, s(u_{l-1})$ la liste des articles déjà placé et $b_1, \cdots, b_k$ la liste des boîtes déjà utilisés et on suppose que la boîte b_i est utilisé qu'à 50%.

1. Soit $s(u_l)$ le nouvel article si $s(u_l) \geq 0.5$. Alors soit on met $s(u_l)$ dans b_i (il n'y a plus de boîte b_i avec un taux de 50%), soit $s(u_l)$ est mis dans une nouvelle boîte (il reste donc la boîte b_i avec un taux a 50%).
2. Si $s(u_l) \leq 0.5$, on peut mettre u_l dans une des boîtes (éventuellement b_i). On peut mettre u_l dans b_i donc b_i n'est plus à 50% où éventuellement dans une autre boîte déjà utilisée, ainsi b_i reste à 50%.

Notons maintenant α le taux de charge de la boîte la moins chargée.

1. Si $\alpha \leq 0.5$ alors $\frac{FF(I)-1}{2} + \alpha < \sum_{i=1}^{n} s(u_i)$ et donc $FF(I) < 2\sum_{i=1}^{n} s(u_i) - 2\alpha + 1$ et donc $FF(I) \leq 2\sum_{i=1}^{n} s(u_i)$.
2. Si $\alpha > 0.5$ alors $\frac{FF(I)}{2} \leq \sum_{i=1}^{n} s(u_i)$ et donc $FF(I) \leq 2\sum_{i=1}^{n} s(u_i)$.

□

2.6.1 La borne est atteinte

En prenant comme instance $s(u_1), s(u_2), \cdots, s(u_n) = (0.5, 1/(2N), 0.5, 1/(2N), 0.5, \ldots, 1/(2N))$. Dans la solution optimale on trouve, $N + 1$ boites et en utilisant l'algorithme nous trouvons $2N$.

Deuxième partie

Introduction des communications

CHAPITRE

3 Le modèle UET-UCT

Sommaire

3.1 Introduction 42
3.2 Problème UET-UCT avec une infinité de processeurs 42
3.2.1 Complexité du problème avec graphe de précédence quelconque 43
3.2.2 Seuil d'approximation pour un graphe quelconque et pour la somme des temps de complétude 46
3.3 Approximation avec garantie de performance non triviale . 48
3.3.1 Le programme linéaire en nombres entiers 49
3.3.2 Ordonnancement réalisable 52
3.3.3 Analyse de la performance relative de l'algorithme . . 54
3.3.4 La borne supérieure de la performance relative 54
3.4 Problèmes avec m processeurs 58
3.4.1 Complexité du problème avec un graphe de précédence quelconque 58
3.4.2 Seuil d'approximation pour le problème de la minimisation de la longueur de l'ordonnancement pour un graphe quelconque 61
3.4.3 Seuil d'approximation pour un graphe quelconque et pour la somme des temps de complétude 65
3.4.4 Complexité dans le cas où le graphe de précédence est un arbre 67
3.5 Approximation pour le problème avec m processeurs . . . 70
3.5.1 Un premier algorithme 70
3.5.2 La borne supérieure est atteinte 73
3.5.3 Approximation : la notion de pliage en ordonnancement 73
3.5.4 Le pliage évolué 76

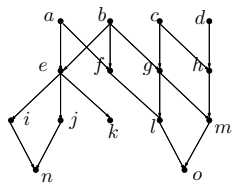
3.1 Introduction

Dans ce chapitre, nous allons nous introduire la notion de communications entre deux tâches adjacentes dans le graphe de précédence. Par la suite, nous allons focaliser notre attention sur les problèmes $UET - UCT$ (Unit Execution Time- Unit Communication Time) avec une infinité de processeurs ($\bar{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$) et un nombre limité ($P|prec, p_i = 1, c_{ij} = 1|C_{max}$). Nous étudions ce deux problèmes "basiques" pour déterminer la complexité intrinsèque de ces problèmes.

Nous avons négligé le temps de transfert des données entre deux tâches qui s'exécutent sur des processeurs différents. Nous avons simplement qu'un arc (i, j) du graphe de précédence est traduit tout simplement par $t_j - t_i \geq p_i$. Afin de prendre le modèle plus réaliste, on introduit les coûts de communications c_{ij} de la façon suivante :

Si (i, j) est un arc du graphe de précédence :
– $t_j - t_i \geq p_i$ si i et j sont exécutées sur le même processeur,
– sinon $t_j - t_i \geq p_j + c_{ij}$

Exemple :



Graphe $UET - UCT$

	0	1	2	3	4	5	6	7
π_0	a				k			
π_1	b							
π_2			e	i				
π_3			f					
π_4	c				j	n		
π_5	d		h		m		o	
π_6			g		l			
π_7					k			

Ordonnancement du graphe

FIG. 3.1 – *Exemple d'ordonnancement sur un système multiprocesseurs*

3.2 Problème UET-UCT avec une infinité de processeurs

Dans cette partie nous allons intéresser au problème basique en ordonnancement, qui consiste à ordonnancer sur une infinité de processeurs un graphe de pré-

cédence quelconque tel que la durée des tâches est unitaire et la communication entre chaque paire de sommets relié par un arc est également unitaire. Ce problème sera par la suite $\bar{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$. Dans un premier temps nous allons nous intéresser à déterminer la complexité de ce problème et ensuite de proposer un algorithme approché avec garantie de performance non triviale. Nous allons regarder si il est plus facile d'ordonnancer sur une infinité de processeurs que sur un nombre limité. Nous allons déterminer pour quel valeur C_{max} associé à la longueur de l'ordonnancement, le problème d'ordonnancer un graphe de précédence quelconque tel que la durée des tâches est unitaire et la communication entre chaque paire de sommets relié par un arc est également unitaire est un problème plus difficile sur un nombre limité de processeurs que sur un nombre illimités.

3.2.1 Complexité du problème avec graphe de précédence quelconque

Nous allons chercher la frontière entre l'existence d'une solution polynomiale et un résultat de non-approximabilité.

Théorème 3.2.1 *Le problème de décider si une instance du problème* $\bar{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$ *possède un ordonnancement de longueur* 5 *est polynomial.*

Preuve

La preuve est basée sur la notion de matrice uni-modulaire.

□

La frontière entre l'existence d'une solution polynomiale et la $\mathcal{NP}-$complétude est donné par le théorème suivant :

Théorème 3.2.2 *Le problème de décider si une instance du problème* $\bar{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$ *possède un ordonnancement de longueur au plus* 6 *est* $\mathcal{NP}-$*complet.*

Preuve

Notre preuve est basée sur la réduction $3SAT \propto \bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$.

Données :

- soit $\mathcal{V} = \{x_1, \ldots, x_n\}$ un ensemble de n variables logiques.
- soit $\mathcal{C} = \{C_1, \ldots, C_m\}$ un ensemble de clauses contenant 3 littéraux $(x_{c_i} \vee y_{c_i} \vee z_{c_i})$.

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 au moins un littéral mis à vrai par I ?

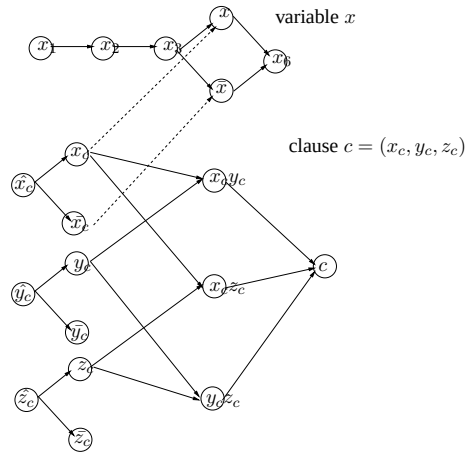


FIG. 3.2 – *Les variables-tâches et les clauses-tâches*

Il est clair que notre problème est dans $\mathcal{NP}$. En effet, une solution du problème d'ordonnancement $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$ consiste à déterminer pour chaque tâche i une date de début d'exécution t_i et un processeur π_i. Il est donc facile de vérifier en temps polynomial que la solution proposée est réalisable ou non en six unités de temps.

Soit π^* une instance quelconque du problème $3SAT$, nous construisons une instance π du problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$ de la manière suivante.

- Pour chaque variable x, nous introduisons six tâches : $x_1,\ x_2,\ x_3,\ x,\ \bar{x}$ and x_6 ; les contraintes de précédence entre ces tâches sont donnés par la figure 3.2.
- Pour chaque clause $c = (x_c, y_c, z_c)$, où les littéraux $x_c,\ y_c$ et z_c représentent des variables qui apparaissent sous forme positive ou négative, nous introduisons 13 variables : $\hat{x_c},\ \hat{y_c},\ \hat{z_c},\ x_c,\ \bar{x_c},\ y_c,\ \bar{y_c},\ z_c,\ \bar{z_c}, x_cy_c,\ y_cz_c,\ x_cz_c$ and c : les contraintes de précédence entre ces tâches sont également données par la figure 3.2.
- Si la variable x dans la clause c est sous forme x, alors x_c précède la variable x et $\bar{x_c}$ précède la variable $\bar{x}$.

– Si la variable x dans la clause c est sous forme $\bar{x}$, alors x_c précède la variable $\bar{x}$ et $\bar{x_c}$ précède la variable x.

Clairement, x_c représente l'occurrence de la variable x dans la clause c, elle précède la variable-tâche correspondante. La construction est illustrée à la figure 3.2. La transformation proposée ci-avant est une transformation calculable en temps polynomial.

- Supposons qu'il existe une affectation $I : \mathcal{V} \rightarrow \{0, 1\}$ des valeurs de vérité aux variables satisfaisant l'ensemble des clauses. Alors un ordonnancement de longueur au plus 6 peut être obtenu de la manière suivante.
  Nous allons commencer par faire deux remarques essentielles :
  – Premièrement, dans un ordonnancement en temps 6 il y a exactement deux manières d'ordonnancer les tâches correspondant à la variable x, sachant que x est exécuté à l'instant 4 et que $\bar{x}$ exécuté à l'instant 5 (ou l'inverse). Dans les deux cas, les tâches $x_1,\ x_2$ et x_3 sont exécutées par le même processeur que la variable qui est ordonnancé à l'instant 4 et la tâche x_6 est exécuté par le même processeur qui exécute la variable-tâche à l'instant 5.
  – Deuxièmement, nous pouvons noter que dans le but d'ordonnancer en 6 unités de temps, les clauses-tâches correspondant à la clause $c = (x_c, y_c, z_c)$, alors au moins une des tâches $x_c,\ y_c$ et z_c doit s'exécuter à l'instant 2. Supposons qu'il existe une affectation des valeurs de vérité qui satisfasse le problème $3SAT$, alors l'ordonnancement de longueur 6 est obtenu de la manière suivante :
    – On exécute la variable-tâche x à l'instant 4 si la **variable x est "vraie"** et à l'instant 5 sinon.
    – Si le littéral x apparaissant dans la clause c est "vrai", alors x_c est exécuté à l'instant 2 sur le même processeur que $\hat{x_c}$, et $\bar{x_c}$ est exécuté à l'instant 3 ;
    – Si le littéral x apparaissant dans la clause c est "faux", alors x_c est exécuté à l'instant 3 et $\bar{x_c}$ est exécuté à l'instant 2 ;
    – Les autres tâches sont exécutées de manière gloutonne.

  Sachant que chaque clause c contient au moins un littéral mis à "vrai", chaque clause-tâche c est complété à l'instant 6.
- Réciproquement, supposons maintenant qu'il existe un ordonnancement de longueur au plus six. 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 $3SAT$.
  Nous définissons une variable x à **vrai** si la variable-tâche x correspondante est exécutée à l'instant 4, et **faux** sinon. Sans perte de généralité, nous supposons qu'une variable x est exécutée à l'instant 4. Chaque occurrence positive de x doit être exécutée à l'instant 2 et chaque occurrence négative de x à l'instant 3, impliquant que tous les littéraux sont affectés de valeurs consistantes (un littéral ne peut avoir de affectation différentes). Sachant que chaque clause-tâche c termine son exécution à $t = 6$, nous savons que chaque clause contient au moins un littéral à vrai.

Il est conseillé de faire la construction avec pour instance pour le problème $3 - SAT$: $(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee x_4)$.

□

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

Théorème 3.2.3 *Le problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|\sum_j C_j$ ne possède pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à $9/8$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve

Nous allons montrer qu'il n'existe pas d'algorithme A garantissant $\frac{C^A_{max}}{C^{opt}_{max}} < \frac{9}{8}$ sauf si $\mathcal{P} = \mathcal{NP}$. Pour obtenir ce résultat, nous allons nous baser d'une part sur la transformation polynomiale utilisée pour la démonstration du théorème 3.2.2, et d'autre part sur la technique du "gap" (voir le théorème 1.2.5). Lors de la construction de l'instance du problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$, à partir du problème $\mathcal{NP}$-complet $3SAT$, l'ordonnancement associé à cette transformation polynomiale utilise un ensemble de M processeurs (M arbitrairement grand) et un ensemble de tâches unitaires soumises à des contraintes de précédence et des délais de communications unitaires qui sont exécutées sur ces processeurs.

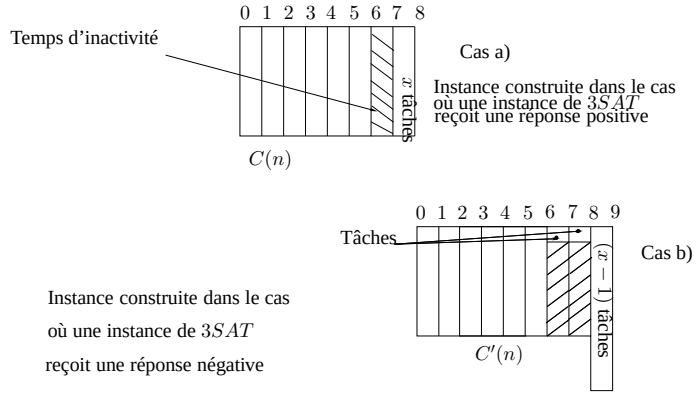


FIG. 3.3 – *Construction de la transformation polynomiale pour la somme des temps de complétude à partir de celle effectuée pour la longueur de l'ordonnancement.*

Il est facile de vérifier que :

- Dans le cas où l'instance du problème $3SAT$ reçoit une réponse positive (c'est-à-dire que nous pouvons conclure à l'existence d'une affectation de valeurs de vérités aux variables telle que chaque clause contienne exactement un littéral mis à vrai), dans l'instance associée au problème d'ordonnancement, les tâches sont exécutées dans l'intervalle $[0, 6]$. Alors la somme des temps de complétude associé est $C(n) \in \mathbb{N}$ où n désigne le nombre de variables dans l'instance du problème $3SAT$.
- Dans le cas où l'instance du problème $3SAT$ reçoit une réponse négative (c'est-à-dire que nous pouvons conclure à la non-existence d'une affectation de valeurs de vérité aux variables telle que chaque clause contienne exactement un littéral mis à vrai), alors dans n'importe quel ordonnancement réalisable au moins (pour l'instance construite) une tâche a une fin d'exécution à 7 ou plus. Alors la somme des temps de complétude est égale à $C'(n) > C(n)$ et $C'(n) \in \mathbb{N}$.

Nous ajoutons aux tâches de l'instance initiale un ensemble de x nouvelles tâches. Dans la relation de précédence, chaque nouvelle tâche est un successeur des anciennes tâches (les anciennes tâches sont issues de la transformation polynomiale utilisée pour la démonstration du théorème 3.2.2). Nous obtenons ainsi un graphe

complet entre les anciennes tâches et les nouvelles tâches. Notons cette instance I^* et observons la propriété suivante :

- Dans le cas où l'instance du problème $3SAT$ reçoit une réponse positive, alors à partir de l'instance I^*, nous construisons un ordonnancement avec la somme des temps de complétude égale à $C(n) + 8x$ (voir la figure 3.3 cas a)). Nous laissons un temps d'inactivité entre les anciennes tâches qui finissent leur exécution à $t = 6$, et les nouvelles tâches pour respecter les délais de communications.
- Dans le cas où l'instance du problème $3SAT$ reçoit une réponse négative, alors à partir de l'instance I^*, la somme des temps de complétude est au moins égale à $C'(n) + 15 + 9(x - 1)$ (voir la figure 3.3 cas b)). En effet, au plus une tâche peut s'exécuter sur le même processeur que la tâche qui a un début d'exécution à l'instant 6.

Donc, en faisant tendre x vers des grandes valeurs, il existe un algorithme d'approximation en temps polynomial avec une garantie de performance strictement inférieure à $9/8$ qui peut être utilisé pour distinguer en temps polynomial les instances positives des instances négatives du problème $3SAT$, fournissant ainsi un algorithme polynomial pour un problème $\mathcal{NP}$-difficile. Par conséquent, le problème $\bar{P}|prec; c_{ij} = 1; p_i = 1| \sum_j C_j$ ne possède pas d'algorithme ρ-approché, avec $\rho < 9/8$.

Autre méthode

Il suffit d'appliquer le Théorème 1.2.5 du Chapitre 1 avec les valeurs $z = 9,\ y = 8, K = 9, k = 6$.

□

3.3 Approximation avec garantie de performance non triviale

Dans cette section, nous allons présenter un algorithme approximation ayant une performance relative de $4/3$ basé sur une formulation par un programme linéaire en nombres entiers et une relaxation des contraintes d'intégrités. La borne de $4/3$ est la meilleure borne connue à ce jour. Un programme linéaire est défini par un ensemble d'inéquations linéaires, appelées contraintes, et par une fonction de coût linéaire ; l'objectif est de minimiser la fonction de coût tout en en satisfaisant les inéquations.

Formellement, soit $G = (V; E)$ un graphe de précédence où V est l'ensemble des tâches (avec $|V| = n$) et E l'ensemble des dépendances entre les tâches de G.

Définition 3.3.1 *Nous désignons par Z (resp. U) l'ensemble des tâches sans prédécesseur (resp. sans successeur). Nous appellerons source toute tâche appartenant à l'ensemble Z.*

Un ordonnancement σ est l'ensemble de n triplets ordonnés $\sigma = \{(i,\ \pi^i,\ t_i), i \in V\}$. Chaque triplet représente une tâche i exécutée par un processeur π^i à l'instant t_i. Tout ordonnancement réalisable doit vérifier les conditions suivantes définies par le vecteur des dates de début d'exécution $(t_1, \ldots, t_n)$:

1. à tous les instants, un processeur exécute au plus une tâche ;
2. $\forall (i,j) \in E$, si $\pi^i = \pi^j$ alors $t_j \geq t_i + p_i$, sinon $t_j \geq t_i + p_i + c_{ij}$.

La longueur de l'ordonnancement σ est égale à la date de fin d'exécution de la dernière tâche du graphe G. Formellement, la longueur d'un ordonnancement σ est définie de la manière suivante :

$$C^{\sigma}_{max} = \max_{i \in V} (t_i + p_i)$$

Le problème est de déterminer un ordonnancement avec une longueur minimale.

Remarque : l'heuristique simple qui consiste à exécuter les tâches dès que possible tout en respectant les contraintes de précédence et les éventuels délais de communications par des clusters différents possède une performance relative de deux.

3.3.1 Le programme linéaire en nombres entiers

Nous proposons une formulation du problème $\bar{P}|prec; c_{ij} = 1; p_i = 1|C_{max}$ comme un programme linéaire en nombres entiers, noté PL_I dans la suite.

Nous modélisons le problème d'ordonnancement par un ensemble d'équations définies par rapport au vecteur des dates de début d'exécution $(t_1, \ldots, t_n)$: dans n'importe quel ordonnancement réalisable chaque tâche $i \in V - U$ a au plus un successeur $j \in \Gamma^+(i)$, qui peut être exécutés sur le même processeur que i au temps $t_j < t_i + p_i + 1$.

Les autres successeurs de i, s'ils existent, satisfont : $\forall k \in \Gamma^+(i) - \{j\},\ t_k \geq t_i + p_i + 1$. Alors, pour chaque arc $(i,j) \in E$, nous introduisons d'une part une variable $x_{ij} \in \{0,1\}$, qui modélise la présence ou non du délai de communication entre les tâches i et j, et d'autre part les conditions suivantes :

$$\forall (i,j) \in E,\ t_i + p_i + x_{ij} \leq t_j$$

et

$$\sum_{j \in \Gamma^+(i)} x_{ij} \geq |\Gamma^+(i)| - 1.$$

De manière similaire, chaque tâche i de $V - Z$ admet au plus un prédécesseur $k \in \Gamma^-(i)$, qui peut être exécutés par le même module que i et satisfait les contraintes : $t_i - (t_k + p_k) < 1$. Ainsi, nous introduisons les contraintes :

$$\sum_{j \in \Gamma^-(i)} x_{ji} \geq |\Gamma^-(i)| - 1.$$

Sachant que C_{max} désigne la longueur de l'ordonnancement, nous en déduisons l'inégalité suivante :

$$\forall i \in V,\ t_i + p_i \leq C_{max}.$$

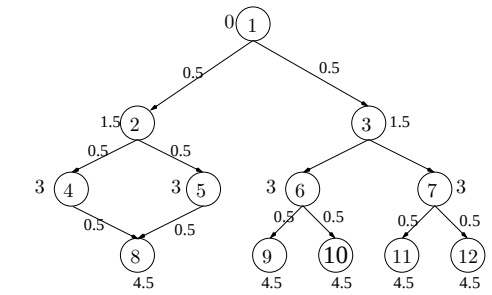
On peut formuler notre problème d'ordonnancement, en un problème de programmation linéaire en nombres entiers en abrégé (P.L.N.E.) :

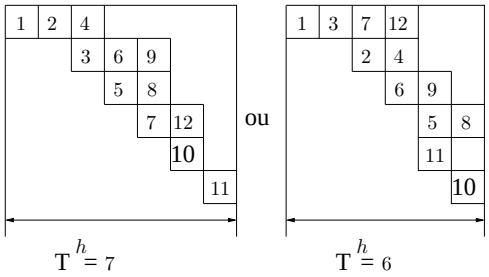
$$PL_I \begin{cases} & minC_{max} \\ \forall (i,j) \in E, & x_{ij} \in \{0,1\} \\ \forall i \in V, & t_i \geq 0 \\ \forall (i,j) \in E, & t_i + 1 + x_{ij} \leq t_j \\ \forall i \in V - U, & \sum_{j \in \Gamma^+(i)} x_{ij} \geq |\Gamma^+(i)| - 1 \\ \forall i \in V - Z, & \sum_{j \in \Gamma^-(i)} x_{ji} \geq |\Gamma^-(i)| - 1 \\ \forall i \in V & t_i + 1 \leq C_{max} \end{cases}$$

Si nous relâchons les contraintes d'intégrité on ne considère plus les variables x_{ij} à valeur dans l'ensemble $\{0,1\}$ mais des variables x_{ij} à valeur dans l'intervalle $[0,1]$, on obtient ainsi un problème de programmation linéaire qui peut être résolu en temps polynômial.

On dénote par $\theta = (\theta_1, \ldots, \theta_n)$ la solution du problème de programmation linéaire et par $e = (\{e_{i,j}, (i,j) \in E\}$ les valeurs correspondantes des variables x_{ij}.

On appelle t^h_i le début d'exécution de la tâche i déterminé par l'heuristique h.





ou

FIG. 3.4 – *Exemple d'ordonnancement avec performance relative de* $\frac{4}{3}$.

Prenons par exemple le graphe donnée par la figure 3.4. La programmation linéaire donne pour solution $\forall (i,j) \in E, e_{i,j} = \frac{1}{2}$,$\theta_1 = 0, \theta_2 = \theta_3 = \frac{3}{2}$, $\theta_4 = \theta_5 = \theta_6 = \theta_7 = 3$,$\theta_8 = \theta_9 = \theta_{10} = \theta_{11} = \frac{9}{2}$ et le temps d'ordonnancement est de $\frac{11}{2}$. Sachant que $e_{1,2} = e_{1,3} = \frac{1}{2}$ les tâches 2 et 3 ont la même priorité, donc si $t_2^h = 1$ on a $C_{max}^h = 7$ et si $t_3^h = 1$ on a $C_{max}^h = 6$.

Soit PL_I^{inf}, le programme linéaire correspondant à PL_I dans lequel nous relaxons les contraintes d'intégrité en transformant $x_{ij} \in \{0,1\}$ en $x_{ij} \in [0,1]$. Sachant que le nombre de variables et le nombre de contraintes sont bornés polynomialement, ce programme linéaire peut-être résolu en temps polynomial. Une solution de PL_I^{inf} affectera à tout arc $(i,j) \in E$ une valeur $x_{ij} = e_{ij}$ avec $0 \leq e_{ij} \leq 1$ et déterminera une borne inférieure de la valeur C_{max} que nous noterons par la suite Θ^{inf}.

Lemme 3.3.1 *Θ^{inf} est une borne inférieure de la solution optimale pour le problème d'ordonnancement $\bar{P}|prec; c_{ij} 1; p_i = 1|C_{max}$.*

Preuve

Ceci est vrai sachant que toute solution optimale réalisable doit satisfaire les contraintes du programme linéaire en nombres entiers PL_I.

□

Algorithme 3.1 *Algorithme d'arrondis*

Étape 1 [**Arrondis**]

Soit e_{ij} l'évaluation de l'arc $(i,j) \in E$ obtenue par la solution PL_I^{inf}

$$\begin{cases} \text{si } e_{ij} < 0.5 & \Longrightarrow \quad x_{ij} = 0 \\ \text{si } e_{ij} \geq 0.5 & \Longrightarrow \quad x_{ij} = 1 \end{cases}$$

Définition 3.3.2 *Étant donnée une solution de PL_I^{inf}, la longueur d'un chemin est définie comme la somme des durées d'exécution des tâches appartenant à ce chemin et des valeurs des variables x_{ij} associées aux arcs (i,j) qui appartiennent à ce chemin.*

Définition 3.3.3 *Un chemin critique de sommet terminal i ($i \notin Z$) est le plus long chemin à partir d'une tâche source de G jusqu'à la tâche i.*

Définition 3.3.4 *Nous appelons un arc $(i,j) \in E$ un 0-arc (resp. 1-arc) si $x_{ij} = 0$ (resp. $x_{ij} = 1$).*

Définition 3.3.5 *A_i est défini comme le sous-ensemble des arcs entrants d'extrémité i appartenant à un chemin critique ayant pour sommet terminal la tâche i.*

3.3.2 Ordonnancement réalisable

Dans cette section, nous allons montrer que l'algorithme proposé ci-dessus produit un ordonnancement réalisable.

Nous donnons ci-dessous deux définitions qui serviront de base à la démonstration du théorème 3.3.1.

Définition 3.3.6 *Nous dirons qu'une tâche i est une tâche totalement libre si tous les arcs entrants, avec pour sommet terminal la tâche i, sont évalués à 1.*

Dans le cas où la tâche $i \in V$ est totalement libre, alors cette tâche peut-être exécutée sur n'importe quel module.

Définition 3.3.7 *Nous dirons qu'une tâche i est une tâche libre par rapport à la tâche j, si $(j,i) \in E$ et l'arc (j,i) est évalué à 1.*

Dans ce cas la tâche i peut être exécutée sur un processeur différent du module sur lequel la tâche j est ordonnancée.

Algorithme 3.2 *Construction de l'ordonnancement*

Étape 1 [**Détermination des dates de début d'exécution**]
si $i \in Z$ **alors**
  $t_i = 0$
sinon
  $t_i = \max\{t_j + p_j = 1 + x_{ji}\}$ avec $j \in \Gamma^-(i)$ et $(j,i) \in A_i$
fin si
Étape 2 [**Construction de l'ordonnancement**]
Soit $G' = (V; E')$ où $E' = E \backslash \{(i,j) \in E | x_{ij} = 1\}$ {G' est le graphe engendré par les $0 - arcs$.}
Affecter chaque composante connexe du graphe G' sur des processeurs différents.
Chaque tâche est exécutée à sa date de début d'exécution.

L'algorithme 3.2 construit l'ordonnancement associé à la solution donnée par l'algorithme 3.1. Il se décompose également en deux étapes.

Dans la première étape de l'algorithme, nous déterminons les dates de début d'exécution des tâches en utilisant la valeur de la variable x_{ij} déterminée par l'algorithme 3.1. Pour cela, il suffit de calculer un chemin critique avec i pour sommet terminal.

Dans la deuxième étape, nous déterminons le placement des tâches sur les modules. Nous considérons par la suite le graphe $G' = (V; E')$ où E' désigne l'ensemble des arcs et qui est défini de la manière suivante : $E' = E \backslash \{(i,j) \in E | x_{ij} = 1\}$. Le graphe G' est le graphe partiel de G engendré par les arcs ayant une évaluation nulle. Ensuite chaque composante connexe du graphe G' est affectée à un module différent et chaque tâche s'exécute à sa date de début d'exécution.

Théorème 3.3.1 *L'algorithme 3.2 produit une solution réalisable.*

Preuve

Il est clair que l'algorithme 3.2 produit un ordonnancement réalisable sachant qu'il y a au plus un couplage entre les tâches s'exécutant à deux instants consécutifs.

Par conséquent, la manière décrite ci-dessus d'exécuter les tâches préserve les contraintes de précédence, respecte les délais de communication et fournit un ordonnancement réalisable.

□

3.3.3 Analyse de la performance relative de l'algorithme

Dans cette partie, nous allons évaluer la performance relative associée à l'heuristique que nous avons développée. Nous montrerons que la borne supérieure de la performance relative est égale à $\frac{4}{3}$, et que cette borne est atteinte pour une classe particulière de graphes.

3.3.4 La borne supérieure de la performance relative

Par définition, nous notons t_i^h la date de début d'exécution de la tâche i déterminée par l'heuristique et t_i^* la date de début d'exécution de la tâche i donnée par la solution du programme linéaire (t_i^* est le plus long chemin d'une source à la tâche i incluant les durées d'exécution des tâches et l'évaluation e_{ij} des arcs correspondants).

À partir des contraintes du programme linéaire, il est facile de voir que le lemme suivant est vérifié.

Lemme 3.3.2 *Chaque tâche $i \in V$ admet au plus un successeur (resp. prédécesseur) tels que $e_{ij} < 0.5$ (resp. $e_{ji} < 0.5$).*

Preuve

Nous considérons une tâche $i \in V$ et ces successeurs $j_1, \ldots, j_k$ indexés de la manière suivante $e_{i,j_1} \leq e_{i,j_2} \leq \ldots \leq e_{i,j_k}$. Nous savons que $\sum_{l=1}^{k} e_{i,j_l} \geq k - 1$ (par la P.L.), alors

$$2e_{i,j_2} \geq e_{i,j_2} + e_{i,j_1} \geq k - 1 - \sum_{l=3}^{k} e_{i,j_l}.$$

sachant que $e_{i,j_l} \in [0,1]$, $\sum_{l=3}^{k} e_{i,j_l} \leq k - 2$. Alors $2e_{i,j_2} \geq 1$. Ainsi $\forall l \in \{2, \ldots, k\}$ nous avons $e_{ij} \geq 0.5$. On montre de même pour les prédécesseurs. □

Lemme 3.3.3 *Pour chaque tâche $i \in V$, $t_i^h \leq \frac{4}{3} t_i^*$.*

Preuve

Nous allons montrer ce résultat par récurrence.

L'inégalité est vraie pour chaque tâche $i \in Z$ (i.e. pour les tâches telles que $t_i^h = 0$) et pour chaque tâche k telle que $\Gamma^-(k) \subseteq Z$.

Supposons maintenant que le lemme est valide pour tous les prédécesseurs de i.

Soit l'ensemble A_i comme définit dans la définition 3.3.5.

Nous devons considérer les cas suivants.

1. Un des arcs de l'ensemble A_i noté (j,i) est évalué à 0 ($x_{ji} = 0$, avec $e_{ji} < 0.5$). Alors la date de début d'exécution donnée par l'heuristique pour la tâche i est $t_i^h = t_j^h + p_j$. De plus, la date de début d'exécution pour la tâche i donnée par le programme linéaire est $t_i^* \geq t_j^* + p_j$.

   Par l'hypothèse de récurrence, nous avons $t_j^h \leq \frac{4}{3} t_j^*$.

   Ainsi, $t_i^h \leq \frac{4}{3} t_j^* + p_j$ et par conséquent $t_i^h \leq \frac{4}{3}(t_i^* - p_j) + p_j \leq \frac{4}{3} t_i^*$.
2. Tous les arcs de l'ensemble A_i noté (j,i), est évalué à 1 ($x_{ji} = 1$) et $e_{ji} \geq 0.5$ 5il n'y a pas de . Alors, $t_i^h = t_j^h + 1 + p_j$, et $t_i^* \geq t_j^* + p_j + 0.5$. Par l'hypothèse de récurrence, nous avons $t_j^h \leq \frac{4}{3} t_j^*$.

   Nous obtenons $t_i^h \leq \frac{4}{3} t_j^* + p_j + 1$ et ainsi $t_i^h \leq \frac{4}{3}(t_i^* - 0.5 - p_j) + 1 + p_j \leq \frac{4}{3} t_i^*$.

□

Finalement nous avons le théorème :

Théorème 3.3.2 *L'algorithme proposé est un algorithme $\frac{4}{3}$-approché.*

Preuve

Soit C_{max}^h la longueur de l'ordonnancement obtenue par notre heuristique et soit C_{max}^{opt} la longueur de l'ordonnancement optimal.

Considérons une tâche i de U telle que $C_{max}^h = t_i^h + p_i$. Par le lemme 3.3.3, nous savons que $C_{max}^h \leq \frac{4}{3}(t_i^* + p_i)$. Par le programme linéaire, nous obtenons que $t_i^* + p_i \leq \Theta^{inf}$ et que $\Theta^{inf} \leq C_{max}^{opt}$ (Θ^{inf} est une borne inférieure de la solution optimale, voir le lemme 3.3.1). Ainsi, le théorème est démontré.

□

3.3.4.1 La borne est atteinte pour une classe particulière de graphes

Dans cette section, nous allons définir une classe particulière de graphes pour laquelle la performance relative de l'algorithme proposé est asymptotiquement égale à $\frac{4}{3}$. Pour ceci, nous définissons les graphes B_1 et B_2 qui serviront de

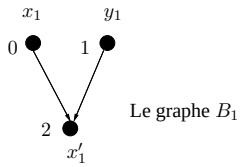


FIG. 3.5 – *Le graphe B_1 et son ordonnancement associé $\sigma(B_1)$.*

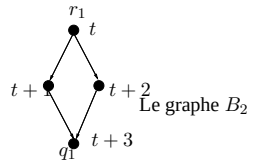


FIG. 3.6 – *Le graphe B_2 et son ordonnancement associé $\sigma(B_2)$.*

base pour notre construction. Nous allons définir récursivement une séquence de graphes $G_i,\ i > 1$, basée sur G_{i-1} et sur B_2.

Nous allons calculer la valeur $C_{max}^h(G_i)$ obtenue par notre heuristique et nous allons construire un ordonnancement réalisable σ tel que :

$$\lim_{i \to \infty} \frac{C_{max}^h(G_i)}{C_{max}^\sigma(G_i)} = \frac{4}{3}.$$

Les graphes B_1, B_2 et leurs ordonnancements associés $\sigma(B_1)$ et $\sigma(B_2)$ sont donnés par les figures 3.5 et 3.6 respectivement.

Notation : par la suite, quand nous écrirons "$B_1 @ B_2$", nous considérons le graphe obtenu par la concaténation du graphe B_1 et du graphe B_2, dans lequel nous agrégeons les tâches du dernier niveau du graphe B_1 avec les tâches du premier niveau du graphe B_2. Plus précisément, $x'_1 = r_1$.

La définition de $G_{i-1} @ B_2$ est faite de manière similaire (les tâches du dernier niveau de G_{i-1} sont agrégées avec les tâches du premier niveau de B_2 i.e. $q_1 = r_1$ avec $q_1 \in V(G_{i-1})$ et $r_1 \in V(B_2)$).

Nous construisons un ensemble de graphes G_i de la manière suivante :

- G_1 est défini comme $B_1 @ B_2$.
- et G_i est défini comme $G_{i-1} @ B_2$ avec $i \geq 2$.

3.3.4.2 La longueur de l'ordonnancement pour les graphes G_i

Lemme 3.3.4 *La longueur de l'ordonnancement pour le graphe B_1 (resp. B_2) donné par l'heuristique est égale à $C_{max}^h(B_1) = 3$ (resp. $C_{max}^h(B_2) = 5$).*

Preuve

La solution du programme linéaire relaxé affectera la valeur 0.5 à tous les arcs. Ainsi, l'heuristique transformera toutes ces valeurs à 1, et par là même la longueur de l'ordonnancement sera égale à 3 pour le graphe B_1 (resp. 5 pour le graphe B_2).

□

Lemme 3.3.5 *La longueur de l'ordonnancement pour le graphe G_i donné par l'heuristique est égale à $C^h_{max}(G_i) = 4i + 3$*

Preuve

Nous procédons par récurrence sur i.

- Si $i = 1$ le lemme est vérifié : $C^h_{max}(G_1) = C^h_{max}(B_1)+C^h_{max}(B_2)-1 = 7$.
- Nous supposons que le lemme est vérifié pour $i-1,\ i \geq 2$ c'est-à-dire $C^h_{max}(G_{i-1}) = 4(i-1)+3$.
  Nous avons $C^h_{max}(G_i) = C^h_{max}(G_{i-1}) + C^h_{max}(B_2) - 1 = 4(i-1)+3+5-1 = 4i+3$.

□

On construit l'ordonnancement σ récursivement.

- $\sigma(G_1)$ est obtenu par concaténation des ordonnancements de B_1 et de B_2 (voir les figures 3.5 et 3.6) en prenant en compte l'agrégation des tâches dans $G_1 = B_1 @ B_2$ ($x'_1 = r_1$).
- De manière similaire, l'ordonnancement $\sigma(G_i)$ est obtenu par concaténation des ordonnancements de G_{i-1} et de B_2 (en prenant en compte les tâches qui ont été agrégées).

Lemme 3.3.6 *La longueur de l'ordonnancement donné par σ pour le graphe G_i, $C^\sigma_{max}(G_i)$, est égale à $3i + 3$.*

Preuve

Par récurrence sur i.

- Si $i = 1$ le lemme est valide ($C^\sigma_{max}(G_1) = 6$).
- Nous supposons que le lemme est vérifié pour $i-1,\ i \geq 2$ c'est-à-dire $C^\sigma_{max}(G_{i-1}) = 3(i-1)+3$. Sachant que $C^\sigma_{max}(B_2) = 4$, nous obtenons $C^h_{max}(G_i) = 3(i-1)+3+4-1 = 3i+3$.

□

Théorème 3.3.3 *La borne $\rho^h = \frac{4}{3}$ est atteinte pour les graphes G_i.*

Preuve

Par les lemmes 3.3.5 et 3.3.6, nous savons que $C^h_{max}(G_i) = 4i+3$ et que $C^\sigma_{max}(G_i) = 3i+3$.

Alors,

$$\lim_{i\to\infty} \frac{C^h_{max}(G_i)}{C^\sigma_{max}(G_i)} = \lim_{i\to\infty} \frac{4i}{3i} = \frac{4}{3}.$$

Par définition de la performance relative pour une heuristique h, nous avons

$$\rho^h = \max_G \frac{C^h_{max}(G)}{C^{opt}_{max}(G)} \geq \lim_{i\to\infty} \frac{C^h_{max}(G_i)}{C^\sigma_{max}(G_i)} = \frac{4}{3}$$

En utilisant le théorème 3.3.2, nous obtenons alors l'égalité c'est-à-dire $\rho^h = \frac{4}{3}$.

□

3.4 Problèmes avec m processeurs

3.4.1 Complexité du problème avec un graphe de précédence quelconque

Dans cette partie, nous allons étudier la complexité du problème $P|prec, p_i = 1, c_{ij} = 1|C_{max}$. Nous allons montrer que le problème précédent est $\mathcal{NP}-$complet. Ceci a été prouvé par Rayward-Smith [28].

Théorème 3.4.1 *Le problème de décider si une instance du problème $P|prec, p_i = 1, c_{ij} = 1|C_{max}$ possède un ordonnancement de longueur C_{max} est $\mathcal{NP}-$complet.*

Preuve

Nous allons reformuler le problème de la manière suivante :

Instance : Un ensemble T de tâches avec un ordre partiel $<$, un nombre $m > 0$ de processeurs et un temps limite b.

Question : Exite-t'il un ordonnancement valide de T sur m processeurs qui autorise des communications unitaires entre deux tâches relié par une relation d'ordre s'exécutant sur des processeurs différents ayant une longueur inférieure ou égale à b ?

Plus formellement, existe-t'il une injection $s^c : T \to \{1,2,\ldots,m\} \times \{1,2,\ldots,b\}$ tels que $\forall a,\ b \in T, a < b$ implique

– soit $\delta_1(s^c(a)) = \delta_1(s^c(b))$ et $\delta_2(s^c(a)) < \delta_2(s^c(b))$

– ou $\delta_1(s^c(a)) \neq \delta_1(s^c(b))$ et $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$?

Ci-dessus δ_1 et δ_2 sont les fonctions de projections standard. Ce problème sera appelé par la suite $SPOUTC$.

Clairement le problème $SPOUTC \in \mathcal{NP}$.

Pour montrer la $\mathcal{NP}-$complétude de $SPOUTC$ nous allons proposer la transformation polynomiale suivante à partir du problème $\mathcal{NP}-$complet $SPOUT$.

Problème SPOUT :

Instance : Un ensemble T de tâches (non indépendantes) avec un ordre partiel $<$, un nombre $T \geq m > 0$ de processeurs et un temps limite $b \leq |T|$.

Question : Exite-t'il un ordonnancement valide de T sur m processeurs ayant une longueur inférieure ou égale à b ?

Plus formellement, existe-t'il une injection $s : T \rightarrow \{1, 2, \ldots, m\} \times \{1, 2, \ldots, b\}$ tels que $\forall a,\ b \in T, a < b$ implique $\delta_2(s(a)) < \delta_2(s(b))$?

Ce problème est $\mathcal{NP}-$complet c'est le Théorème de Ullman [30].

Soit $I = (T, <, m, b)$ une instance de $SPOUT$. Nous allons construire une instance $f(I)$ de $SPOUTC$ de la manière suivante :

- Aux tâches de T, nous allons ajouter un ensemble V de tâches aux nombres de $(b+1)(m+1)+b$. Les contraintes de précédene entre ces tâches sont les suivantes : $\forall 0 \leq i < b$ et $\forall 1 < j \leq m+1,\ a_{ij}$ est parent de $a_{(i+1)1}$ et a_{i1} est parent de chaque $a_{(i+1)j}$. De plus,, $\forall 0 \leq i \leq b,\ a_{i1}$ est parent de c_{i+1} qui lui-même est parent de $a_{(i+1)1}$.
- Le nombre de processeurs est $m+1$ et $C_{max} = 2b+1$.

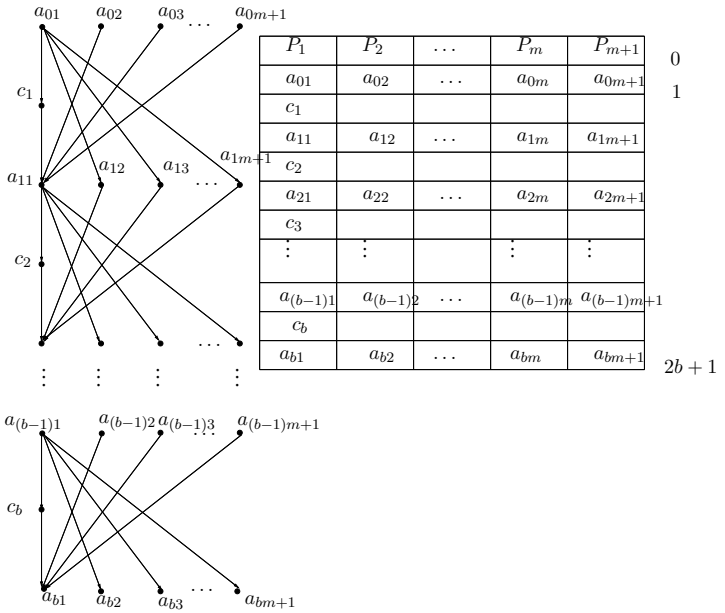


FIG. 3.7 – *Les tâches V, et l'ordonnancement optimal*

La construction est illustrée par la figure 3.7.

- Supposons que I admet une réponse positive, alors il existe une injection $s : T \rightarrow \{1, 2, \ldots, b\}$ tel que $\forall a,\ b \in T,\ a < b$ implique $\delta_2(s(a)) < \delta_2(s(b))$. Nous pouvons définir $s^c : T \rightarrow \{1, 2, \ldots, m+1\} \times \{1, 2, \ldots, 2b+1\}$ par $s^c(a) = (\delta_1(s(a)) + 1, 2\delta_2(s(a)))$. Alors s^c est clairement injective et $\forall a,\ b \in T,\ a < b$ implique $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$. Maintenant, nous étendons s^c en une injection $s^c : T \cup V \rightarrow \{1, 2, \ldots, m+1\} \times \{1, 2, \ldots, 2b+1\}$ par $s(a_{ij}) = (j, 2i+1)$ et $s^c(c_i) = (1, 2i)$. Alors, il est facile de vérifier que $\forall a,\ b \in T \cup V,\ a < b$ implique
  – soit $\delta_1(s^c(a)) = \delta_1(s^c(b))$ et $\delta_2(s^c(a)) < \delta_2(s^c(b))$
  – ou $\delta_1(s^c(a)) \neq \delta_1(s^c(b))$ et $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$.
  Ainsi, $f(I)$ admet une réponse positive pour le problème $SPOUTC$.

- Réciproquement, si $f(I)$ admet une réponse positive pour le problème $SPOUTC$, alors il existe une injection $s^c : T \cup V \rightarrow \{1, 2, \ldots, m+1\} \times \{1, 2, \ldots, 2b+1\}$ tels que $\forall a,\ b \in T \cup V, a < b$ implique
  – soit $\delta_1(s^c(a)) = \delta_1(s^c(b))$ et $\delta_2(s^c(a)) < \delta_2(s^c(b))$
  – ou $\delta_1(s^c(a)) \neq \delta_1(s^c(b))$ et $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$?
  Avec les tâches V, nous avons une chaîne de $2b+1$ tâches, $a_{01} < c_1 < a_{11} < c_2 < \ldots < a_{(b-1)1} < c_b < a_{b1}$. Ainsi, nous pouvons déduire que $\delta_1(s^c(a_{01})) = \delta_1(s^c(c_1)) = \ldots = \delta_1(s^c(a_{b1}))$ et $\delta_2(s^c(a_{i1})) = 2i+1,\ \delta_2(s^c(c_i)) = 2i$. Alors, un processeur est complètement dédié à l'exécution des ces tâches. Sans perte de généralité, nous pouvons supposé que ce processeur dédié est le numéro 1.
  Maintenant, $\forall 0 \leq i < b-1,\ 1 < j \leq m+1$, nous avons également une chaîne $a_{i1} < a_{(i+1)j} < a_{(i+2)1}$. La tâche $a_{(i+1)j}$ ne peut s'exécuter sur le processeur 1. Ainsi, nous déduire que $\delta_2(s^c(a_{(i+1)j})) = 2i+3$. Alors, à l'instant $2i+1,\ 1 \leq i \leq b$, tous les processeurs sont dédiés à l'exécution des tâches $a_{i1,\,i2}, \ldots, a_{im+1}$. Ce résultat reste valide pour $i = 0$ sachant que $a_{01},\ a_{02}, \ldots,\ a_{0m+1}$ doivent être exécutées à l'instant 1. Alors, nous pouvons déduire que $\forall a \in T,\ \delta_1(s^c(a)) \in \{2, 3, \ldots, m+1\}$ et $\delta_2(s^c(a)) \in \{2, 4, \ldots, 2b\}$. Maintenant nous définissons $s(a) = (\delta_1(s^c(a)) - 1, \delta_2(s^c(a))/2)$. $s : T \rightarrow \{1, \ldots, m\} \times \{1, \ldots, b\}$ est injective et satisfait le fait que si $a < b$ alors $\delta_(s(a)) < \delta_2(s(b))$ comme voulu.

□

3.4.2 Seuil d'approximation pour le problème de la minimisation de la longueur de l'ordonnancement pour un graphe quelconque

Dans cette partie nous nous intéressons aux problèmes où le nombre de ressources est limité ce qui induit une difficulté supplémentaire. Dans un premier temps, nous allons nous intéressé à déterminer la complexité et dans un deuxième temps, nous proposerons deux algorithmes approchées avec garantie de performance qui se basent sur la solution donnée sur une infinité de processeurs.

Théorème 3.4.2 *Le problème de décider si une instance de $P|prec, c_{ij} = 1, p_i = 1|C_{max}$ possède un ordonnancement de longueur au plus 3 est polynomial.*

Preuve La preuve est donné dans la thèse de Picouleau [26]. □

Théorème 3.4.3 *Le problème de décider si une instance de $P|prec, c_{ij} = 1, p_i = 1|C_{max}$ possède un ordonnancement de longueur au plus 4 est $\mathcal{NP}$-complet.*

Preuve

La démonstration sera basé sur le problème **Clique** :
Données : un graphe non orienté $G = (V, E)$ et un entier k.
Question : Existe-il une clique de taille k dans G ?

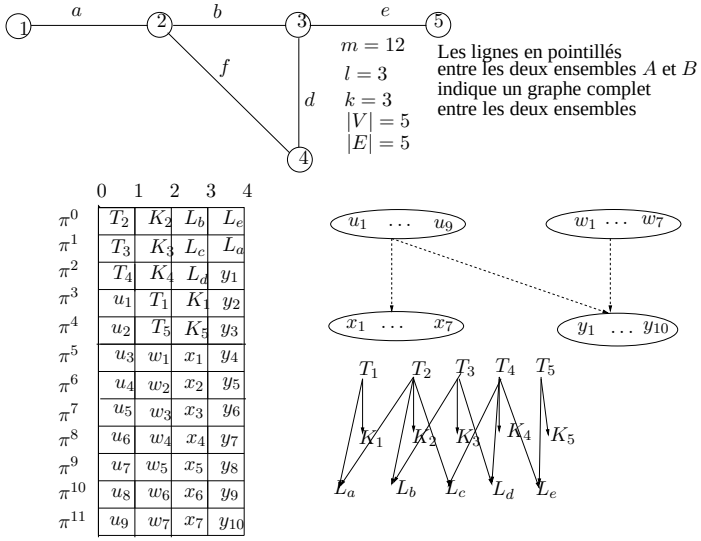


FIG. 3.8 – *Exemple de transformation polynomiale* $clique \propto P|prec; c_{ij} = 1; p_i = 1|C_{max}$.

On note $l = \frac{k(k-1)}{2}$ le nombre d'arêtes d'une clique de taille k. On pose $m' = \max\{|V| + l - k, |E| - l\}$. Le nombre de processeurs de l'instance est

$m = 2(m' + 1)$. Il est clair que notre problème est dans $\mathcal{NP}$. Notre preuve est basée sur la réduction $clique \propto P|prec, c_{ij} = 1, p_i = 1|C_{max}$. Soit une instance π^* quelconque du problème $clique$. Nous construisons une instance π du problème $P|prec, c_{ij} = 1, p_i = 1|C_{max}$ de la manière suivante :

- $\forall v \in V$ on introduit deux tâches $T_v,\ K_v$,
- $\forall e \in E$ un sommet L_e.
  - Nous créons les contraintes de précédence suivantes : $T_v \rightarrow K_v, \forall v \in V$ et $T_v \rightarrow L_e$ si v est une extrémité de e.
- On introduit également 4 ensembles des tâches :
  - $X_x = \{x = 1, \ldots, x = m - l - |V| + k\}$,
  - $Y_y = \{y = 1, \ldots, y = m - |E| + l\}$,
  - $U_u = \{u = 1, \ldots, u = m - k\}$,
  - $W_w = \{w = 1, \ldots, w = m - |V|\}$.
  - Nous créons les contraintes de précédence suivantes : $U_u \rightarrow X_x,\ U_u \rightarrow Y_y,\ W_w \rightarrow Y_y$

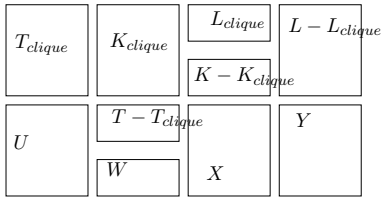


FIG. 3.9 – *Exemple de construction pour illustrer la preuve du théorème 3.4.3*

Prenons l'exemple donné par la figure 3.9.

- Supposons que G possède une clique de taille k, alors il existe un ordonnancement de longueur 4. il suffit d'ordonnancer les tâches en se basant sur le diagramme de Gantt donné par la figure 3.9.
- Supposons qu'on a un ordonnancement valide de longueur 4 unités de temps alors G contient une clique de taille K.
  1. Toutes les tâches U_u doivent être exécutées à l'instant 1 : supposons que ceci n'est pas vrai. Alors au moins $|X|+|Y|-1$ tâches doivent être exécutées à l'instant 4. On a $|X|+|Y|-1 = 2m+k-(|E|+|V|+1) > m$.
  2. Toutes les tâches de Y_y doivent être exécutées à l'instant 4. Pour pouvoir exécuter des tâches de Y_y avant 4, il faut que U et W soient exé-

cutes à $t = 1$. Or $|U| + |W| > m$. Or d'après 1 il y a $m - k$ processeurs bloqués pour l'exécution de U.

3. Au plus k tâches de type T peuvent être exécutées à l'instant 1 et par conséquent au plus l tâches de type L peuvent être exécutées à l'instant 3.

Si on exécute moins de k tâches à l'instant 1, au moins de (l_L) tâches à l'instant 3, l'ordonnancement dans $'$ unités de temps ne serait pas réalisable. Puisque $|E| - l'(l' < l)$ tâches restantes doivent être exécutées à l'instant 4. Or le nombre de processeurs disponibles est $|E| - l < |E| - l'$. En effet, si on prend $l' < l \Rightarrow |E| - l' > |E| - l$. Or $|E| - l$ =nombre de processeurs disponible implique que $|E| - l'$ demande un instant supplémentaire. Ceci est possible si G contient une clique de taille k.

□

Corollaire 3.4.1 *Pour le problème $P|prec, c_{ij} = 1, p_i = 1|C_{max}$, il n'existe pas d'algorithme approché avec une garantie de performance inférieure à $5/4$ sous l'hypothèse que $\mathcal{P} \neq \mathcal{NP}$.*

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

Théorème 3.4.4 *Le problème $P|prec; c_{ij} = 1; p_i = 1|\sum C_j$ ne possèdent pas d'algorithme d'approximation avec une performance relative garantie strictement inférieure à $11/10$ sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.*

Preuve

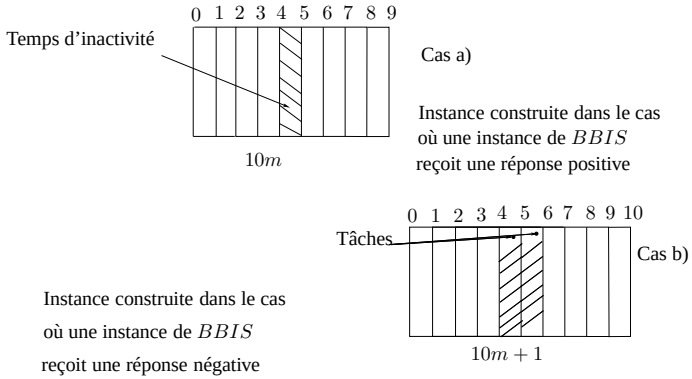


FIG. 3.10 – *Construction de la transformation polynomiale pour la somme des temps de complétude à partir de celle effectuée pour la longueur de l'ordonnancement.*

La démonstration est similaire à celle faite pour le théorème 2.5.2.

Pour obtenir ce résultat, nous considérons la transformation polynomiale utilisée pour la démonstration du théorème 2.5.2. Cette transformation utilise un ensemble de m machines (m est une entrée du problème) et un ensemble de tâches unitaires soumises à des contraintes de précédence et des délais de communications unitaires.

Nous pouvons remarquer que :

– Dans le cas où l'instance de $BBIS$ reçoit une réponse positive (c'est-à-dire nous pouvons conclure à l'existence d'un stable équilibré d'ordre k dans le graphe de précédence), dans l'instance associée au problème d'ordonnancement, les $8m$ tâches sont exécutées dans l'intervalle $[0, 4]$. Ainsi, la somme des temps de complétude est de $10m$ unités de temps.

– Dans le cas où l'instance de $BBIS$ reçoit une réponse négative (c'est-à-dire nous pouvons conclure à la non-existence d'un stable équilibré d'ordre k dans le graphe précédence), dans n'importe quel ordonnancement réalisable au moins (pour l'instance construite) une tâche a une fin d'exécution à 4 ou plus. Ainsi, la somme des temps de complétude est au moins de $10m + 1$ unités de temps.

Nous ajoutons aux $8m$ tâches de l'instance initiale un ensemble de $4m$ nouvelles tâches. Dans la relation de précédence, chaque nouvelle tâche est un suc-

cesseur des anciennes tâches. Nous obtenons ainsi un graphe complet entre les anciennes tâches et les nouvelles tâches.

Notons cette instance I^* et observons la propriété suivante :

- Dans le cas où l'instance de $BBIS$ reçoit une réponse positive, alors à partir de l'instance I^*, nous construisons un ordonnancement avec la somme des fins d'exécution des tâches égale à $40m$ (voir la figure 3.10 cas a)). Nous laissons un temps d'inactivité entre les anciennes tâches qui finissent leur exécution à $t = 4$, et les nouvelles tâches pour respecter les délais de communications.
- Dans le cas où l'instance de $BBIS$ reçoit une réponse négative, alors à partir de l'instance I^*, la somme des fins d'exécution des tâches est au moins égale à $44m + 1$ (voir la figure 3.10 cas b)). En effet, au moins une tâche s'exécute sur un module à $t = 4$. Ainsi il y a plus 1 tâche qui peuvent s'exécuter à l'instant suivant sur le même processeur que la tâche qui a un début d'exécution à l'instant 4. Ainsi la somme des temps de complétude est au moins égale à $10m + 1 + 6 + 7m + 8m + 9m + 10(m - 1) = 44m - 3$.

Donc il existe un algorithme d'approximation en temps polynomial avec une garantie de performance strictement inférieure à $11/10$ qui peut être utilisé pour distinguer en temps polynomial les instances positives des instances négatives du problème $BBIS$, fournissant ainsi un algorithme polynomial pour un problème $\mathcal{NP}$-difficile. Par conséquent, les problèmes $P|prec; c_{ij} = 1; p_i = 1|\sum C_j$ et $P|prec; c_{ij} = 1; p_i = 1; dup|\sum C_j$ ne possèdent pas d'algorithme ρ-approché, avec $\rho < 11/10$.

□

3.4.4 Complexité dans le cas où le graphe de précédence est un arbre

Dans cette partie, nous allons nous intéresser au problème où le cas le graphe de précédence est un arbre. Nous allons montrer que même dans ce cas de figure le problème est $\mathcal{NP}-$complet. IL est important de noter que pour le problème noté $P|tree, p_j = 1|C_{max}$ Hu propose un algorithme linéaire $O(n)$: il utilise un algorithme d'ordonnancement du chemin critique, la prochaine tâche choisie c'est celle en tête de la plus longue chaîne courante des tâches non encore exécutées. Dans la suite, nous allons étudier la complexité du problème $P|tree, p_i = 1, c_{ij} = 1|C_{max}$.

Théorème 3.4.5 *Le problème de décider si une instance de $P|tree, p_i = 1, c_{ij} = 1|C_{max}$ possède un ordonnancement un longueur au plus C_{max} est $\mathcal{NP}-$complet.*

Preuve

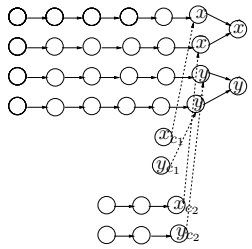


FIG. 3.11 – *Variables-tâches et clauses-tâches correspondant à une instance du problème Satisfaisabilité avec $c_1 = (x \vee \bar{y})$ et $c_2 = (\bar{x}, y)$*

La démonstration sera basé sur le problème **SAT** :

Données : Soit un ensemble U de variables et une collection C de clauses constituée de variables de U.

Question : Existe-il une affectation de variables qui satisfasse les clauses de C.

Soit une instance (U, C) du problème Satisfaisabilité, nous définissons un seuil de valeur $b = 2|C| + 4$ et soit le nombre de processeurs est donné par $m = 2|U| + \sum_{c \in C} |c| + 1$.

- Pour chaque variable x nous introduisons une variable-tâche $\hat{x}$ et deux chaînes constituées de $b - 2$ variables-tâches chacune ; une correspond au littéral x et l'autre au littéral $\bar{x}$. Ces deux chaînes précédent la variable-tâche $\hat{x}$, comme sur la figure 3.11.
- Soit $c_1, \ldots, c_{|c|}$ un ordre arbitraire sur les clauses de C. Pour chaque clause $c_i, (i = 1, \ldots, |C|)$ nous introduisons $|c_i|$ chaînes de variables-clauses constitué chacune de $2i - 1$ tâches ; il y a une correspondance une à une entre les ces chaînes et les littéraux constituant c_i.
- Nous introduisons maintenant les contraintes de précédence entre les variables-tâches et les variables-clauses de la manière suivante :
  - Si l'occurrence de la variable x dans la clause de la forme positive (de la forme x), alors la dernière tâche x_{c_i} de la chaîne-clause correspondant à cette occurrence précède la dernière tâche de la chaîne-variable correspondant au littéral x.
  - Si l'occurrence de la variable x dans la clause c_i est sous forme négative (de la forme $\bar{x}$), alors x_{c_i} précède la dernière tâche de la chaînes de variables correspondant à $\bar{x}$.

– En final, nous introduisons un total de $b+|U|+\sum_{i=1}^{|C|}\{|c_i|(b-2i-3)+1\}$ dummy tâches formés de chaînes :
  – Premièrement, une chaîne de longueur b.
  – Deuxièment, $|U|$ chaînes unitaires constitué d'une tâche unique.
  – Troisièment, pour chaque clause c_i il y a $|c_i|$ chaînes de dummy tâches, une de longueur $b-2i-2$ et les autres de longueur $b-2i-3$. Pour chaîne de dummy de longueur $l, l < b$, la dernière tâche précède la $(l+2)$ ème tâche de la chaîne de dummy tâches de longueur b. Ainsi, dans n'importe quel ordonnancement réalisable de longueur b chaque dummy chaîne sera exécutée sur un processeur unique, la première tâche de chaque chaîne sera exécutée à l'instant $t=0$, et les autres tâches s'exécuteront à la suite sans interruption.

- Supposons une affectation des variables qui satisfasse une instance de (U,C). Montrons qu'il existe un ordonnancement de longueur $C_{max}=b$.
  Soit une telle affectation, nous pouvons construire un ordonnancement de longueur $C_{max}=b$ de la manière suivante :
  – Si la variable x est varie, alors la variable-tâche x est exécutée à l'instant $b-2$ sur le même processeur que la variable-tâche $\hat{x}$ et la variable-tâche $\bar{x}$ est exécutée à l'instant $b-3$.
  – Si la variable x est fausse, alors la variable-tâche $\bar{x}$ est ordonnancé à l'instant $b-2$ sur le même processeur que la variable-tâche $\hat{x}$ et la variable-tâche x est exécutée à l'instant $b-3$.
  – Si x_{c_i} représente l'occurrence positive de x dans la clause c_i et si la variable x est vraie, alors x_{c_i} est exécutée à l'instant $b-4$. Maintenant les tâches correspondantes à la clause c_i sont ordonnancées sur les mêmes processeurs que les chaînes de longueur $b-2i-2$ et $b-2i-3$ de dummy tâches.

  Ainsi, avec un telle affectation de variable, nous pouvons facilement construire un ordonnancement réalisable de longueur $C_{max}=b$.
- Supposons un ordonnancement de longueur $C_{max}=b$, nous allons montrer qu'il existe une affectation des variables qui satisfasse les clauses.
  Les dummy tâches imposent une structure d'ordonnancement donné par la figure 3.12 (les dummy tâches données par des points noirs).
  Au plus $|U|$ non dummy tâches peuvent être exécutées à l'instant $t=0$. Soit une variable x, au moins une des deux chaînes correspondant à x commencent son exécution à $t=0$. Ainsi, pour chaque variable x exactement une de ces chaînes est exécutée dans la période $[0, b-2]$. La dernière chaîne est exécutée par le même processeur que $\hat{x}$, le littéral associé à cette chaîne sera considéré comme vrai.

Soit une clause c_i, la chaîne-clause correspondant peut être exécutée par le processeur qui exécute la dummy chaîne de longueur au plus $b-2i-2$. Il s'ensuit que la chaîne-clause c_i est exécutée par le processeur qui ordonnance la dummy chaîne de longueur $b-2i-2$ et les $|c_i|-1$ processeurs autres exécutent les les dummy chaînes de longueur $b-2i-3$.
Au moins une de ces chaînes-clauses terminent son exécution à l'instant $b-3$, il précède un littéral vrai, sachant que dans le cas contraire l'ordonnancement ne serait pas valide.
Donc, à partir de l'ordonnancement nous pouvons déduire une affectation consistante des variables.

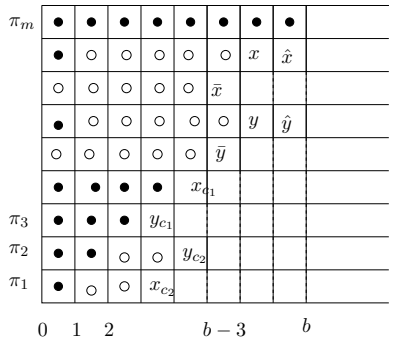


FIG. 3.12 – *Ordonnancement de longueur b*

□

3.5 Approximation pour le problème avec m processeurs

Dans cette partie, nous allons proposer plusieurs algorithmes approchés avec des performances relatives décroissantes pour le problème $P|prec, p_i=1, c_{ij}=1|C_{max}$. Ils sont basés sur plusieurs approches (listes, pliages, ...)

3.5.1 Un premier algorithme

Nous allons proposé le premier algorithme, donné par Rayward-Smith [28] ; qui est basé sur le principe d'un ordonnancement valide, ordonnancement pour le-

quel aucune tâche ne peut commencer plus tôt sans augmenter le début d'exécution d'une autre tâche. Nous utilisons un algorithme de liste.

Théorème 3.5.1 *IL existe un $3 - 2/m$-algorithme approché pour le problème $P|prec, p_i = 1, c_{ij} = 1|C_{max}$.*

Preuve

Nous allons définir un graphe de couche du type (n, m).

Définition 3.5.1 *Un graphe de couche du type (n, m) est un graphe orienté tels que :*

- *si il existe n couches,*
- *tous les sommets terminaux sont à la profondeur n,*
- *Pour n'importe quelle tâche appartenant une couche m $0 \leq m \leq n-1$ alors cette tâche admet au moins un parent.*

Un exemple de graphe de couche de $(5, 2)$ est donné par la figure 3.13.

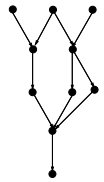


FIG. 3.13 – *Un graphe de couche de type $(5, 2)$*

Lemme 3.5.1 *Soit un graphe de couche de type (n, m) alors l'ordonnancement optimal est d'au moins de $n + m$ unités de temps.*

Preuve

La preuve se fait par récurrence sur n. □

Dans un premier temps, montrer le lemme suivant :

Considérons maintenant un graphe avec des tâches unitaires et soit $C^{h'}_{max}$ la longueur de l'ordonanncement en utilisant un algorihtme glouton sur m' processeurs. Soit $c_1 < c_2 < \ldots < c_r$ les instants dans l'ordonnancement admettant au moins un temps d'inactivité sur au moins un processeur. Si c est un temps d'inactivité, alors soit :

- c est un instant dormant, c'est à dire aucun processdeur n'est actif. Dans ce cas, chaque tâche exécutées après c admet au moins deux tâches prédécesseurs exécutées à $t = c - 1$, ou
- si c n'est pas un instant dormant, il y a au moins un processeur actif. Dans ce cas, chaque tâche exécutée après c admet au moins un prédécesseur exécutée à l'instant c ou à $c - 1$.

Soit $c_{\lambda 1} < c_{\lambda 2} < \ldots < c_{\lambda s}$ les instants dormants. Considérons maintenant la séquences des instants suivants : $c_{\lambda 1} - 1 < c_{\lambda 1} < c_{\lambda 2} - 1 < c_{\lambda 2} < \ldots < c_{\lambda s} - 1 < c_{\lambda s} < C^{h'}_{max}$.

Soit C le temps de cette séquence. Par l'algorithme glouton, nous savons que nous n'avons pas deux instants dormant consécutifs, ne peut commencer et finir par un instant dormant.

Nous allons construire un graphe de couche en partant d'un sommet terminal s'exécutant à $C^{h'}_{max}$. Cette tâche admet au moins deu xprédécesseurs s'écutant à $c_{\lambda s} - 1$. En continuant , ces deux tâches admets deux prédécesseurs à l'instant $c_{\lambda(s-1)} - 1$. En procédant de la même manière, nous pouvons construire un grpahe de couche de type $(s + 1, s)$.

Maintenant considérons les instants C' apparaissant dans la séquence $c_1 < c_2 < \ldots < c_r$ mais n'apparaissant pas dans $c_{\lambda 1} - 1 < c_{\lambda 1} < c_{\lambda 2} - 1 < c_{\lambda 2} < \ldots < c_{\lambda s} - 1 < c_{\lambda s} < C^{h'}_{max}$. Il en exsite au moins $(r - (2s + 1)$.

Prenons $\lceil (r - (2s + 1)/2 \rceil$ instants dans C' tel que si l'instant c est choisit alors $c - 1$ n'est pas sélectionné. Ce nouvel ensemble d'instant est notée par C''. $\forall c \in C''$, nous pouvons ajouter un graphe de couche sachant que pour chaque tâche ordonnancée après $> c$, nous savons qu'il existe un prédécesseur exécutée à $t = c$ ou $t = c - 1$.

Ainsi, nous obtenons un graphe de couche de type $(s + \lceil (r - (2s+1)/2 \rceil + 1, s)$.

Par le lemme 3.5.1, nous obtenons $C^{h'}_{max} \geq 2s + 1 + (r - 2s - 1)/2$.

$$
\begin{aligned}
m' \times C^{h'}_{max} &= \text{temps d'inactivité} + \text{temps activité} \\
&\leq m's + (m' - 1)(r - s) + mC^{h}_{max} \\
&\leq m's + (m' - 1)(2C^{h'}_{max} - 3s - 1) + mC^{h}_{max} \\
&= s(3 - 2m') + C^{h}_{max}(2m' + m - 2) - m' + 1
\end{aligned}
$$

Ainsi, $C^{h'}_{max} \leq (2 + (m - 2)/m')C^{h}_{max} - (2 - 3/m')s - (1 - 1/m')$.

sachant que $s \leq 0$ et $m' > 1$, $(2-3/m')s \geq 0$, nous pouvons établir le résultat en posant $m = m'$ et $C^{h}_{max} = C^{opt}_{max}$. □

3.5.2 La borne supérieure est atteinte

Lemme 3.5.2 *La borne de $3 - 2/m$ est atteinte.*

Preuve

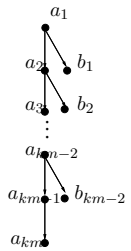


FIG. 3.14 – *Graphe donnant la borne supérieure*

Pour cela il suffit de considérer le graphe donné par la figure 3.14.

– Dans l'ordonnancement optimal des tâches issus du graphe donné par la figure 3.14 sur $m \geq 2$ processeurs exécutent sur un processeurs les tâches a_i et les b_j sur un autre. Le $C_{max}^{opt} = km$.
– Une exécution de l'algorithme peut conduire à exxécutées alternativement les a_i et les b_j sur un seul processeur. Le $C_{max} = 2km - 2$.

Maintenant, rajoutons $km^2 - 2km + 2$ tâches indépendantes. Sur m, l'ordonnancement optimal prendra $C_{max}^{opt} = km$.

Le pire ordonnancement exécutera les tâches indépendantes dans un premier temps et après les autres à partir de $km - 2k + 1$. Si nous alternons entre les a_i et les b_j, nous obtenons $C_{max} = km - 2k + 2km - 2 = 3km - 2k - 2$. En faisant tendre k vers l'infini, nous avons le rapport désiré. □

3.5.3 Approximation : la notion de pliage en ordonnancement

Dans cette partie nous nous intéressons au passage d'un ordonnancement sur une infinité de processeurs à un ordonnancement sur un nombre fixé de machines. Le principe consiste à partir de l'ordonnancement sur une infinité de processeurs et de plier cet ordonnancement afin de respecter les contraintes de ressources.

3.5.3.1 Le pliage simple

Algorithme 3.3 Ordonnancement sur m à partir d'un ordonnancement sur une infinité de processeurs : première méthode

pour $i = 0$ à $C_{max}^{\infty} - 1$ **faire**
Soit X_i est l'ensemble des tâches exécutées à l'instant i avec l'heuristique h^*.
Nous exécutons les tâches de X_i en $\lceil \frac{|X_i|}{m} \rceil$ instants.
fin pour

Théorème 3.5.2 *A partir de n'importe quelle heuristique de rapport de performance ρ pour le problème $\bar{P}|prec, c_{ij} = 1, p_i = 1|C_{max}$ on peut obtenir une heuristique pour le problème $P|prec, c_{ij} = 1, p_i = 1|C_{max}$ de rapport de performance $1 + \rho$.*

Preuve

On appelle h^* l'heuristique utilisé pour le problème $\bar{P}|prec, c_{ij} = 1, p_i = 1|C_{max}$ et h pour le problème $P|prec, c_{ij} = 1, p_i = 1|C_{max}$. Pour h on exécute les tâches dans le même ordre que pour h^*. Si X_i est l'ensemble des tâches exécutées à l'instant i avec l'heuristique h^*, et β_i le cardinal de X_i, alors dans h, si $\beta \neq 0$, on exécute les tâches de X_i en $\lceil \frac{\beta_i}{m} \rceil$ instants où m est le nombre de modules. Dans le cas où $\beta_i = 0$ on conserve l'instant d'inactivité.

On remarque que le nombre d'instants où il y a au moins un module inactif est inférieur à la durée de l'ordonnancement avec un nombre illimité de modules. Les autres instants, tous les modules sont actifs. Ainsi l'écart entre le temps donné par l'heuristique et le temps optimal est borné par la durée de l'ordonnancement avec un nombre illimité de modules. Si l'on note t^* la durée de l'ordonnancement donné par h^* et t^m la durée de l'ordonnancement donné par h alors $t^m \leq t^* + t_{opt}^m$. Comme par hypothèse $t^* \leq \rho t_{opt}^*$ et comme $t_{opt}^* \leq t_{opt}^m$, on obtient le résultat

cherché.

$$
\begin{aligned}
C^m_{max} &\leq \sum_{i=0}^{C^\infty_{max}-1} \lceil \frac{|X_i|}{m} \rceil \\
C^m_{max} &\leq \sum_{i=0}^{C^\infty_{max}-1} (\lfloor \frac{|X_i|}{m} \rfloor + 1) \\
C^m_{max} &\leq \sum_{i=0}^{C^\infty_{max}-1} (\lfloor \frac{|X_i|}{m} \rfloor) + C^\infty_{max} \\
C^m_{max} &\leq \sum_{i=0}^{C^\infty_{max}-1} (\frac{|X_i|}{m}) + C^\infty_{max} \\
C^m_{max} &\leq C^{opt,m}_{max} + C^\infty_{max} \\
C^m_{max} &\leq C^{opt,m}_{max} + \frac{4}{3} C^{opt,\infty}_{max} \\
C^m_{max} &\leq C^{opt,m}_{max} + \frac{4}{3} C^{opt,m}_{max} \\
C^m_{max} &\leq \frac{7}{3} C^{opt,m}_{max}
\end{aligned}
$$

□

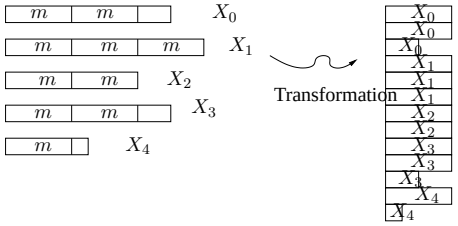


FIG. 3.15 – *Illustration de la preuve du théorème 3.5.2*

La figure 3.15 donne le fonctionnement de l'algorithme 3.3.

Corollaire 3.5.1 *La performance relative associée à la première méthode est de* $\frac{7}{3}$.

Preuve

La meilleure borne connue pour le problème sur un infinité de processeurs est de $\frac{4}{3}$. □

3.5.4 Le pliage évolué

Dans cette partie nous allons proposer un nouvel algorithme de pliage qui utilise la notion de fils favoris et qui produit un algorithme avec une performance relative de $\frac{7}{3} - \frac{4}{3m}$.

Définition 3.5.2 *Nous appelons i fils favoris de j si $t^\infty_i = t^\infty_j + 1$. Ainsi dans σ^∞ l'ordonnancement sur une infinité de processeurs les tâches i et j s'exécutent consécutivement sur le même processeur. Le fils favoris s'il existe sera noté par la suite par $f(i)$.*

Voici algorithme qui permet de construire l'ordonnancement σ^m sur un nombre limité de processeurs en se basant sur σ^∞. Nous supposons que l'ordonnancement partiel a été construit dans l'intervalle $[0, t[$.

Algorithme 3.4 Algorithme qui donne les date d'exécution des tâches

Soit $F = F_1 \bigcup F_2$ l'ensemble des tâches qui n'a pas été encore ordonnancé.
F_1 est l'ensemble des tâches ayant tous ses prédécesseurs ayant fini leurs exécution à t
Soit une tâche j qui a été ordonnancé à $t-1$, nous notons par $S(j)$ l'ensemble des tâches successeurs de j qui peuvent être ordonnancer à t sur le même processeur que $j : i \in S(j) \leftrightarrow \forall k \in \Gamma^-(i) - \{j\}$, k est ordonnancé à $t-1$. Nous pouvons noter que deux tâches de $S(j)$ ne peuvent être exécutées à t. Si le fils favoris $j, f(j) \in S(j)$ alors $f(j) \in F_2$, sinon F_2 contient une tâche arbitraire de $S(j)$.
tant que $F \neq \varnothing$ **faire**
  Nous choisissons un ensemble $F' = \min(|F_1 \bigcup F_2|, m)$ tâches de $F_1 \bigcup F_2$ ordonnançables à t.
  Les tâches de $F' \bigcap F_2$.
fin tant que

Nous notons par $\mathcal{I}[t, t+1[$ le nombre de processeurs inactifs durant l'intervalle $[t, t+1[$. Par extension nous notons $\mathcal{I}[t, t'[= \sum_{\alpha=t}^{t'-1} \mathcal{I}[\alpha, \alpha+1[$. Pour chaque tâche i avec $\Gamma^-(i) \neq \varnothing$ nous notons également par $p(i)$ le dernier prédécesseur de i avec la date d'exécution maximale dans l'ordonnancement σ^m. Si i admet plusieurs prédécesseurs exécutés à t alors nous choisissons pour $p(i)$ celui pour lequel la valeur $t^\infty_{p(i)}$ est minimum.

Lemme 3.5.3 *Pour n'importe quelle tâche i telle que $t^m = t^m_{p(i)} + 1$ alors $\mathcal{I}[t^m_{p(i)}, t^m_i[\leq (m-1)(t^\infty_i - t^\infty_{p(i)})$.*

Preuve

Le nombre de processeurs inactifs à $t_{p(i)}$ dans σ^m est borné par $m-1$. De plus, sachant que $p(i)$ est un prédécesseur de i nous avons $t^\infty_i - t^\infty_i \geq 1$ et on obtient l'inégalité. □

Lemme 3.5.4 *Pour n'importe quelle tâche i telle que $t^m \geq t^m_{p(i)} + 2$ alors $\mathcal{I}[t^m_{p(i)}, t^m_i[\leq (m-1)(t^\infty_i - t^\infty_{p(i)})$.*

Preuve

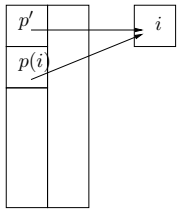


FIG. 3.16 – *Illustration de la preuve 1*

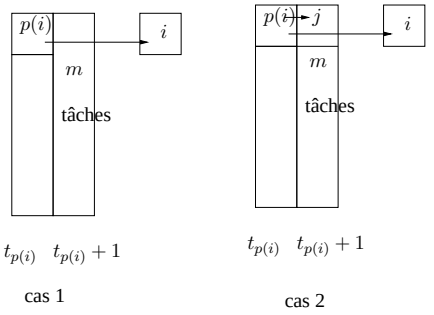


FIG. 3.17 – *Illustration de la preuve 2*

Sachant que $t^m \geq t^m_{p(i)} + 2$, l'interval de temps $[t^m_{p(i)}, t^m_i[$ peut-être décomposé en sous intervalle $\mathcal{I}[t^m_{p(i)}, t^m_i[= \mathcal{I}[t^m_{p(i)}, t^m_{p(i)} + 2[+ \mathcal{I}[t^m_{p(i)} + 2, t^m_i[$.

- Nous allons montrer dans un premier temps que $\mathcal{I}[t^m_{p(i)} + 2, t^m_i[= 0$. En effet, à chaque instant $t \in \{t^m_{p(i)} + 2, \ldots, t^m_i - 1\}$ la tâche i est ordonnaçable sur n'importe quel processeur. Alors à l'étape t de l'algorithme $i \in F_1$. Ainsi, s'il y avait un processeur de libre à l'instant t alors toutes les tâches de F aurait dû être ordonnancé, contradiction.
- Alors $\mathcal{I}[t^m_{p(i)}, t^m_i[= \mathcal{I}[t^m_{p(i)}, t^m_{p(i)} + 2[$. Nous allons prouver que $\mathcal{I}[t^m_{p(i)}, t^m_{p(i)} + 2[\leq (m-1)(t^\infty_i - t^\infty_{p(i)})$ en considérant deux cas :
  1. Si i admet un autre prédécesseur p' exécuté à l'instant $t^m_{p(i)}$, alors il y a au plus m processeurs inactifs à l'instant $t^m_{p(i)} + 1$ et $m-2$ processeurs inactifs à l'instant $t^m_{p(i)}$ (voir figure 3.16). De plus sachant que $t^\infty_{p'} \geq t^\infty_{p(i)}$ par définition de $p(i)$ alors nous avons $t^\infty_i - t^\infty_{p(i)} \geq 2$, et l'inégalité est démontrée.
  2. Si i admet seulement un prédécesseur exécuté à l'instant $t^m_{p(i)}$, alors i est ordonnancé à l'instant $t^m_{p(i)} + 1$. Nous devons étudier deux sous-cas (voir figure 3.17) :
     (a) S'il n'existe pas de processeur inactifs à l'instant $t^m_{p(i)} + 1$, alors il y a au plus $m-1$ processeurs inactifs à l'instant $t^m_{p(i)}$. Alors $t^\infty_i - t^\infty_{p(i)} \geq 1$ et l'inégalité est prouvée.
     (b) Sinon, sachant que i n'est pas exécutée à l'instant $t^m_{p(i)} + 1$, il y a une tâche j exécutée à $t^m_{p(i)} + 1$ qui est successeur de $p(i)$. Alors,

i ne peut être le successeur favoris de $p(i)$ et donc $t_i^\infty - t_{p(i)}^\infty \geq 2$. De plus, il y a au plus $m-1$ processeurs inactifs à l'instant $t_{p(i)}^m+1$ et $m-1$ à $t_{p(i)}^m$, et l'inégalité est prouvée.

□

Lemme 3.5.5 *Pour n'importe quelle tâche i alors $\mathcal{I}[0, t_i^m[\leq (m-1)t_i^\infty$.*

Preuve Nous allons prouver ce résultat par induction sur la longueur $l(i)$ (c'est à dire sur le nombre d'arcs) d'un chemin partant d'un sommet sans prédécesseur à i.

- Si $l(i) = 0$ alors i n'admet pas de prédécesseur dans G, alors $\mathcal{I}[0, t_i^m[= 0$. et l'inégalité est vraie.
- Nous supposons que l'inégalité est vrai pour n'importe quelle tâche telle que $l(i) \leq k,\ k \geq 0$. Nous considérons une tâche i avec $l(i) = k+1$. Alors, $\mathcal{I}[0, t_i^m[= \mathcal{I}[0, t_{p(i)}^m[+\mathcal{I}[t_{p(i)}^m, t_i^m[$. En utilisant les lemmes 3.5.3 et 3.5.4, nous avons $\mathcal{I}[t_{p(i)}^m, t_i^m[\leq (m-1)(t_i^\infty - t_{p(i)}i^\infty)$ et par induction $\mathcal{I}[0, t_i^m[\leq (m-1)t_{p(i)}^\infty$. Le lemme est démontrée.

□

Théorème 3.5.3 *La longueur de l'ordonnancement sur m processeurs en utilisant le pliage du fils favoris est $C_{max}^m \leq \frac{n}{m} + (1-\frac{1}{m})C_{max}^\infty$.*

Preuve

L'ordonnancement σ^m satisfait l'égalité suivante : $mC_{max}^m = n + \mathcal{I}[0, C_{max}^m[$. Maintenant considérons une tâche i exécutée à l'instant $t_i^m = C_{max}^m - 1$. Alors nous obtenons $\mathcal{I}[0, C_{max}^m[= \mathcal{I}[0, t_i^m[+\mathcal{I}[t_i^m, C_{max}^m[$.

Par le lemme 3.5.5 nous avons $\mathcal{I}[0, t_i^m[\leq (m-1)t_i^\infty \leq (m-1)(C_{max}^\infty - 1)$.

De plus, il y a au plus $(m-1)$ processeurs inactifs durant l'intervalle $[C_{max}^m - 1, C_{max}^m[$, ainsi $\mathcal{I}[0, C_{max}^m[\leq (m-1)C_{max}^\infty$ et $mC_{max}^m \leq n + (m-1)C_{max}^\infty$. □

Théorème 3.5.4 *La longueur de l'ordonnancement sur m processeurs en utilisant le pliage du fils favoris admet pour performance relative de $\frac{7}{3} - \frac{4}{3m}$.*

Preuve

Sachant que $\frac{n}{m} \leq C_{max}^{opt,m}$ et $C_{max}^\infty \leq \rho C_{max}^{\infty,opt}$ et par le théorème 3.5.3, nous obtenons

$$C_{max}^m \leq \frac{n}{m} + (1-\frac{1}{m})C_{max}^\infty \leq C_{max}^{opt,m} + \rho(1-\frac{1}{m})C_{max}^{\infty,opt}$$

.

Et sachant que $C_{max}^{opt,m} \geq C_{max}^{\infty,opt}$, nous obtenons l'égalité avec $\rho = \frac{4}{3}$.

□

3.5.4.1 La borne est atteinte

Dans cette partie nous allons montrer l'existence d'un graphe pour lequel la borne donnée par le théorème 3.5.4 est atteinte. Pour cela nous allons construire un graphe $G^m(n)n > 0$ de la manière suivante :

- $G^m(0)$ admet deux sommets a_0 et $a-1$ et un arc (a_0, a_1) ;
- $G^m(n), n > 0$ est obtenu à partir de $G^m(n-1)$ en ajoutant les sommets $a_n,\ b_n,\ c_n, d_n$ et $e_n^1, \ldots, e_n^m$ et les arcs suivants : $(a_n, b_n), (a_n, c_n), (b_n, a_{n-1}), (c_n, a_{n-1})$ et (c_n, d_n) et $\forall \alpha \in \{1, \ldots, m\}, (b_n, e_n^\alpha)$ et (c_n, e_n^α).

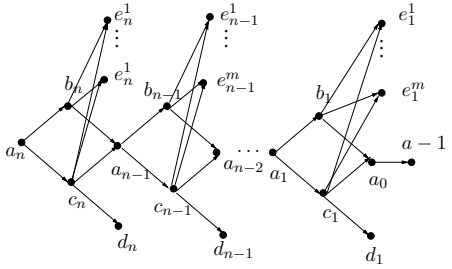


FIG. 3.18 – *Le graphe $G^m(n)n > 0$*

La construction est donné par la figure 3.18.

Soit $H(n)$ le sous-graphe de $G^m(n)n > 0$ constitué seulement des sommets du type a et b.

Pour tous les arcs du graphe $H(n)$ nous avons $\alpha_{ij} = 1/2$ et pour tous les autres arcs de $G^m(n)$ $\alpha_{ij} = 1$ sauf pour l'arc $(a_0, a-1)$, nous avons $\alpha_{0,a-1} = 0$. La longueur de l'ordonnancement est donc pour $H(n) = 3n+2$ sur une infinité de processeurs. l'ordonnancement sur une infinité de processeurs admet pour valeur $x_{a_0 a-1} = 1$ et $xij = 1$ pour tous les autres arcs. Alors dans $G^m(n)$, toutes les tâches autres que a_0 admettent aucun fils favoris.

Maintenant considérons une instance de σ^m donné par le graphe $G^m(n) > 0$ avec $n \mod m = 0$ et $(2m-4)n$ tâches indépendantes. Nous allons construire deux ordonnancements $\sigma_1^m(n)$ et $\sigma_2^m(n)$ donné par la figure 3.19.

- L'ordonnancement σ_2^m commence par exécutées les $(2m-4)n$ tâches indépendantes et continue avec l'ordonnancement $\sigma_2^m(n)$ et admet pour longueur $C_{max}^2 = \frac{(2m-4)n}{m} + 5n + 2$
- L'ordonnancement σ_1^m admet pour longueur $C_{max}^1 = 3n+3$

Nous pouvons en déduire que que $\frac{C^2_{max}}{C^1_{max}} \geq \frac{\frac{(2m-4)n}{m}+5n+2}{3n+3}$

Ainsi $\lim_{n\to\infty} \frac{C^2_{max}}{C^1_{max}} \geq \frac{7}{4} - \frac{4}{3m}$.

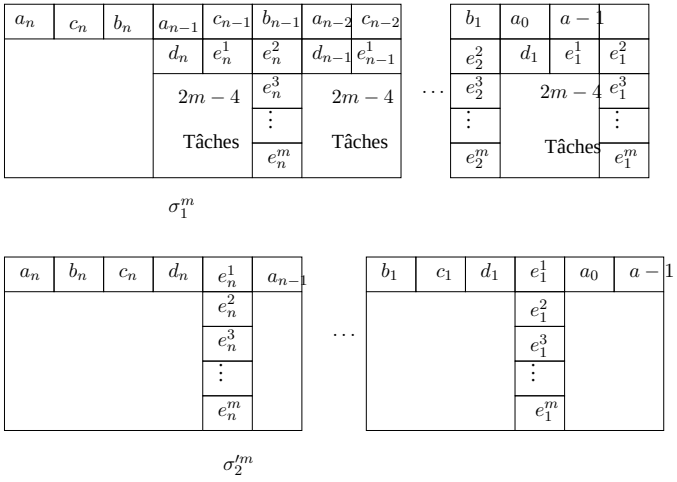


FIG. 3.19 – *Les deux ordonnancements associés au graphe* $G^m(n) n > 0$

CHAPITRE

4 Le modèle SCT

Sommaire

4.1 Introduction 83
4.1.1 Les petits délais de communication 83
4.1.2 Introduction de la duplication 84
4.2 Le problème avec duplication 85
4.2.1 L'algorithme de Colin-Chrétienne 85
4.2.2 Justification et complexité de l'algorithme 86
4.3 Le problème sans duplication 89
4.3.1 Les petits délais de communications 89
4.3.2 Les petits délais de communication locaux 91
4.4 Conclusion 93

4.1 Introduction

4.1.1 Les petits délais de communication

Nous supposons dans cette partie que le graphe de précédence satisfait la condition suivante :

$$\forall j \in V, \quad \min_{i \in PRED(j)} \{p_i\} \geq \max_{i \in PRED(j)} \{c_{ij}\}$$

Ces problèmes interviennent souvent en analyse numérique, les délais de communications sont nettement inférieurs au temps de calcul des tâches (inversion matrices creuses, factorisation de matrices creuses ...).

4.1.2 Introduction de la duplication

La duplication des tâches a été introduite par Papadimitriou et Yannakakis [25] afin de réduire l'influence des délais de communications sur la durée de l'ordonnancement. Dans le cas du modèle avec *communications homogènes*, lorsque le graphe de précédence est soumis aux petits délais de communications locaux, c'est-à-dire dans le cas où les contraintes entre les durées des tâches et les délais de communications sont données par la relation suivante :

$$\forall j \in V, \quad \min_{i \in \Gamma^-(j)} \{p_i\} \geq \max_{i \in \Gamma^-(j)} \{c_{ij}\} \qquad (*)$$

et pour une infinité de processeurs, Colin et Chrétienne [7] ont proposé un algorithme polynomial de complexité quadratique pour résoudre le problème de la minimisation de la longueur de l'ordonnancement. Rappelons ici, que dans le cas où la duplication des tâches n'est pas autorisée, le problème est $\mathcal{NP}$-difficile, même pour le cas UET-UCT avec une infinité de processeurs.

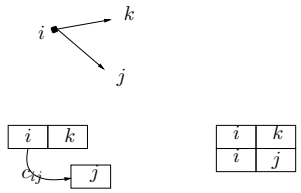


FIG. 4.1 – *Exemple de l'influence de la duplication sur la longueur de l'ordonnancement*.

Le résultat principal pour ce problème a été montré par Ph. Chrétienne et J.Y. Colin dans [7]. Nous allons présenter cet algorithme polynomial en $\bigcirc(|E|)$.

4.2 Le problème avec duplication

4.2.1 L'algorithme de Colin-Chrétienne

4.2.1.1 Présentation et exemple

L'algorithme se déroule en deux parties ; la première calcule les bornes inférieures des dates d'exécution des copies de chaque tâche et la deuxième utilise un arbre critique pour construire un ordonnancement dans lequel toute copie de chacune des tâches est exécutée à sa borne inférieure.

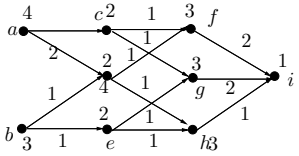


FIG. 4.2 – *Problème P_0.*

Le problème noté P_0 représenté par la figure 4.2 servira d'illustration.

L'algorithme qui détermine les bornes inférieures des dates d'exécutions peut se formuler de la manière suivante :

Algorithme 4.1 Algorithme qui détermine les bornes inférieures d'exécutions, RT

pour $i := 1$ à n **faire**
  si $PRED(i) = \varnothing$ **alors**
    $b_i := 0$
  sinon
    $C := max\{b_k + p_k + c_{ki} \mid k \in PRED(i)\}$;
    soit s tel que : $b_s + p_s + c_{si} = C$;
    $b_i := max\{b_s + p_s, max\{b_k + p_k + c_{ki} \mid k \in PRED(i) - \{s\}\}\}$.
  fin si
fin pour

4.2.2 Justification et complexité de l'algorithme

4.2.2.1 Justification de l'algorithme

Les bornes inférieures pour le problème P_0 sont reportées dans la figure 4.3.

tâches	a	b	c	d	e	f	g	h	i
Borne inf	0	0	4	4	3	7	6	6	11

FIG. 4.3 – *Les bornes inférieures pour P_0.*

Nous allons montrer que la procédure RT calcule la borne inférieure des dates de début d'exécution pour toutes les copies de chaque tâche.

Lemme 4.2.1 *Soit b_i la date de début d'exécution calculée par la procédure RT. Pour n'importe quel ordonnancement réalisable du graphe G la date de début d'exécution des copies de i ne peut pas être inférieure à b_i.*

Preuve

Nous allons procéder par induction sur la profondeur du graphe. Nous allons montrer que si $b_k,\ k \in Pred(i)$ est la borne inférieure de la date de début d'exécution des copies d'une tâche k, alors b_i est la borne inférieure de chaque copie de la tâche i. Cette proposition est trivialement vrai pour les tâches sans prédécesseur. De plus, nous allons démontrer que si (j, i) est un arc de G_c (où G_c désigne le graphe critique associé au graphe G) et $t_j = b_j$ alors $t_i = b_i$ et tous les processeurs qui commencent une copie de j à la date b_j peuvent exécuter la copie de i à la date b_i.

Le lemme est vrai pour les tâches sans prédécesseur $t_i \geq b_i = 0$.

Soit S un ordonnancement, i une tâche et soit c la copie de la tâche i dans S, et considérons la valeur C , la tâche s et le nombre b_i défini par l'algorithme 4.1. Pour tout prédécesseur $k \in PRED(i)$, il existe une copie $c_k \in S$ qui respecte les délais de communications.

- Si c et c_s ne sont exécutées sur le même processeur π, alors nous avons $t_c \leq t_{c_s} + p_s + c_{si} \leq b_s + p_s + c_{si} = C \leq b_i$.
- Si c et c_s sont exécutées sur le même processeur π. Nous avons $t_c \leq t_{c_s} + p_s \leq b_s + p_s$ par l'algorithme 4.1. Soit $k \in PRED(i)\backslash\{s\}$. Si c_k n'est pas ordonnancé sur le processeur π alors nous avons $t_c \leq t_{c_k} + p_k + c_{ki} \leq b_k + p_k + c_{ki}$. Si c_k est exécutée sur le processeur π alors c_k est exécutée

soit avant c_s sur π (avec l'hypothèse des petits délais de communications), ainsi $t_c \leq t_{c_k} + p_k + p_s \leq b_k + p_k + c_{ki}$, ou l'inverse (la dernière relation est encore vraie, en effet, il suffit d'utiliser la définition de s, nous avons $t_c \leq t_{c_s} + p_s + p_k \leq b_s + p_s + c_{si} \leq b_k + p_k + c_{ki}$).

Nous pouvons conclure que $t_c \leq \max\{b_s + p_s, \max\{b_k + p_k + c_{ki} | k \in PRED(i) - \{s\}\}\}$.

Ainsi, pour chaque i la date de début d'exécution de chaque copie de la tâche i ne peut pas être inférieure à b_i et par conséquent, la valeur calculée par la procédure RT est une borne inférieure de la date de début d'exécution de la tâche i.

□

Sans perte de généralité toutes les copies d'une tâche $i \in V$ ont une date de début d'exécution identique à celle de la tâche i notée t_i.

Après avoir déterminé les dates au plus tôt, on utilise un arc critique. Nous dirons qu'un arc $(i, j) \in E$ est critique si $b_i + p_i + c_{ij} > b_j$. A partir de cette définition, il est évident que si (i, j) est un arc critique, alors dans tout ordonnancement au plus tôt toute copie de la tâche i doit être succédé par une copie de la tâche j exécutée sur le même processeur.

Lemme 4.2.2 *Le sous-graphe critique est une forêt couvrante.*

Preuve A partir de la définition de la borne inférieure, nous pouvons conclure qu'il existe au plus une tâche $k \in PRED(l)$ satisfaisant $b_k + p_k + c_{kl} > b_l$. □

Un chemin du graphe de précédence est critique si tous les arcs sont critiques et s'il n'est pas un sous-chemin strict d'un chemin critique. Le sous-graphe critique $\Gamma(\Pi)$ du problème d'ordonnancement Π est le sous-graphe de G induit par les arcs critiques. La propriété fondamentale du graphe critique est que c'est une forêt couvrante. La figure 4.4 représente le sous-graphe critique du problème P_0.

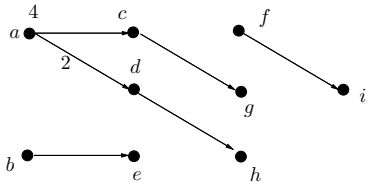


FIG. 4.4 – *Le sous-graphe critique de P_0.*

Pour construire l'ordonnancement au plus tôt sur n processeurs, on alloue un

processeur pour chaque chemin critique et on exécute les copies correspondantes au dates au plus tôt ce qui peut s'écrire :

Algorithme 4.2 *Construction de l'ordonnancement optimal (Earliest Schedule)*

Affecter chaque sous-graphe critique de $G_c = (V; E_c)$ sur des modules distincts ;
Exécuter chaque copie des tâches à sa date au plus tôt.

Théorème 4.2.1 *L'ordonnancement proposé par la procédure ES est réalisable.*

Preuve

Soit i_c une copie de la tâche i et $C^f = \{[f], \ldots, i, \ldots, r\}$ le *sous-graphe critique* de G_c correspondant à la tâche terminale f. Considérons que π est le processeur sur lequel on exécute i_c.

- Si $i = r$ ($Pred_G(i) = \varnothing$) , alors pour chaque tâche k dans $Pred_G(i)$, si elle existe, chaque copie k_c est affectée sur un processeur différent que π par la procédure ES. Sachant que (k, i) n'est pas un arc critique, nous avons $b_i \geq b_k + p_k + c_{ki}$.
- Supposons maintenant que $i \neq r$,et $i \neq f$.
  Soit k une tâche de $Pred_G(i)$ n'appartenant pas au *sous-graphe critique* C_f. Toute copie k_c est affectée à un processeur distinct de π, alors de même nous avons $b_i \geq b_k + p_k + c_{ki}$ sachant que $(k,\ i)$ n'est pas un arc critique.
- Dans le cas où $i = f$, alors avec les mêmes arguments que précédemment, si $k \in Pred_G(i)$ et k n'appartient pas au *sous-graphe critique* C^f, nous avons $b_{i^1} \geq b_k + p_k + c_{ki}$.
- Considérons maintenant k un prédécesseur de i appartenant au *sous-graphe critique* C_f, alors du fait de la procédure ES, une copie k_c est affectée sur le processeur π et elle est exécutée avant i à b_k.

□

4.2.2.2 Analyse de la complexité

Lemme 4.2.3 *Le temps d'exécution de la procédure RT est égal à $\max(n, m)$, où n est le nombre de tâches du graphe de précédence G et m le nombre d'arcs.*

Preuve

Considérons une tâche $i \in V$ dont tous les prédécesseurs ont été examinés. Le calcul de sa date au plus tôt se fait en en $O(PRED(i)$.Par conséquent, la complexité de RT est en $\max(n, m)$

□

Lemme 4.2.4 *La complexité de ES est en $O(n^2)$.*

Preuve

Clairement, la complexité de ES est égal à $O(n^*)$, où n^* dénote le nombre de tâches exécutées (en comptant également les copies). Sachant ceci, il existe au plus $O(n)$ tâches terminales dans le graphe critique G_c (les tâches terminales ne sont jamais dupliquées), utilisant donc au plus $O(n)$ processeurs.

Sachant que chaque processeur n'exécute jamais deux copies de la même tâche ainsi, $n^* < n^2$. Alors, la complexité de ES est $O(n^2)$.

□

Des deux derniers lemmes nous pouvons conclure par le théorème suivant :

Théorème 4.2.2 *La complexité globale de l'algorithme est égale à $O(n^2)$.*

L'ordonnancement obtenu pour le problème P_0 est reporté dans le figure [4.5].

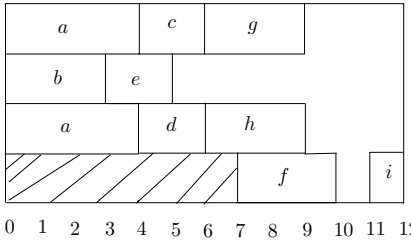


FIG. 4.5 – *L'ordonnancement au plus tôt de P_0.*

4.3 Le problème sans duplication

4.3.1 Les petits délais de communications

Dans cette partie nous allons nous intéresser au problème $\bar{P}|prec, SCT, |C_{max}$. Nous proposons une extension du programme linéaire en nombres entiers proposé dans la section 3.3.1.

On pose

$$\rho = \frac{\max_{(i,j)\in E} c_{ij}}{\min_{i=1,\dots,n} p_i}$$

et nous supposons que $\rho \leq 1$.

$$PL_I \begin{cases} & minC_{max} \\ \forall (i,j) \in E, & x_{ij} \in \{0,1\} \\ \forall i \in V, & t_i \geq 0 \\ \forall (i,j) \in E, & t_i + 1 + x_{ij}c_{ij} \leq t_j \\ \forall i \in V - U, & \sum_{j\in\Gamma^+(i)} x_{ij} \geq |\Gamma^+(i)| - 1 \\ \forall i \in V - Z, & \sum_{j\in\Gamma^-(i)} x_{ji} \geq |\Gamma^-(i)| - 1 \\ \forall i \in V & t_i + p_i \leq C_{max} \end{cases}$$

Notation Ainsi pour x_{ij} fixé, l'ordonnancement est un potentiel sur le graphe G, les arcs admettent un poids $z(i,j) = p_i + x_{ij}c_{ij}$.

Si nous relaxons les contraintes entières $x_{ij} \in \{0,1\}$ en prenant $x_{ij} \in [0,1]$, nous obtenons un programme linéaire noté LP_{inf}, qui peut être résolu en temps polynomial. Notons par C^{inf}_{max} la longueur de l'ordonnancement donné par la résolution du programme linéaire LP_{inf} et x^{inf}_{ij} les valeurs des variables x_{ij} dans cette solution. En utilisant la notation précédente, C^{inf}_{max} est égal au poids maximum sur un chemin de $(G, z^{inf}(i,j))$, pour n'importe quel arc (i,j) et $z^{inf}(i,j) = x^{inf}_{ij}c_{ij}$. Nous obtenons facilement le lemme suivant :

Lemme 4.3.1 *Chaque tâche $i \in V$ admet au plus un successeur (resp. prédécesseur) tels que $x^{inf}ij < 0.5$ (resp. $x^{inf}_{ji} < 0.5$).*

Preuve La preuve est identique à la preuve proposée pour le Lemme 3.3.2. □

Nous utilisons le même algorithme d'arrondis que celui proposé (voir l'algorithme 3.1).

Théorème 4.3.1 *Le performance relative de l'algorithme h basé sur la relaxation d'un programme linéaire en nombres entiers (avec phase d'arrondis) associé au problème $\bar{P}|prec, SCT, |C_{max}$ est $\frac{C^h_{max}}{C_{opt}} \leq \frac{2+2\rho}{2+\rho}$.*

Preuve

D'après le lemme 4.3.1, nous pouvons déduire que chaque arc (i,j) que $z(i,j) \leq \frac{2+2\rho}{2+\rho} z^{inf}(i,j)$. En effet,

– si $x_{ij} = 0$, nous avons $z(i,j) \leq z^{inf}(i,j)$

– sinon ($x_{ij} = 1$), sachant que $x_{ij}^{inf} \geq 1/2$,
nous avons $\frac{z(i,j)}{z^{inf}(i,j)} \leq \frac{p_i + c_{ij}}{p_i + 0.5c_{ij}} \leq 2 - \frac{p_i}{p_i + 0.5c_{ij}}$.
Maintenant $\frac{p_i}{p_i + 0.5c_{ij}} \geq \frac{1}{1+0.5\rho}$ sachant que $\frac{c_{ij}}{p_i} \leq \rho$.
Alors, pour n'importe quel chemin μ de G nous avons $z(\mu) \leq \frac{2+2\rho}{2+\rho} z^{inf}(\mu)$.
□

4.3.2 Les petits délais de communication locaux

Dans cette partie nous allons étendre les résultats précédent dans le cas des petits délias de communications locaux, c'est à dire nous sommes dans les conditions de l'algorithme de Colin et Chrétienne.

Nous rappelons que $C_j = p_j + t_j$. Nous allons étendre le programme linéaire en nombre entiers précédent

$$PL_I \begin{cases} & min C_{max} \\ \forall (i,j) \in E, & x_{ij} \in \{0,1\} \\ \forall i \in V, & C_i \geq p_i \\ \forall (i,j) \in E, & C_i + p_j + (1 - x_{ij}) c_{ij} \leq C_j \\ \forall i \in V - U, & \sum_{j \in \Gamma^+(i)} x_{ij} \geq |\Gamma^+(i)| - 1 \\ \forall i \in V - Z, & \sum_{j \in \Gamma^-(i)} x_{ji} \geq |\Gamma^-(i)| - 1 \\ \forall i \in V & C_i \leq C_{max} \end{cases}$$

Soit (C^{LP}, x^{LP}) une solution du programme linéaire, il est facile d'utiliser une phase d'arrondis pour garantir un ordonnancement réalisable. Le début d'exécution des tâche est donné par l'algorithme suivant :

Algorithme 4.3 Algorithme donnant les dates de début d'exécution

si $\Gamma^-(v) = \varnothing$ **alors**
  $t_v := 0$
sinon
  $t_v := \max\{C_u + (1 - x_{uv}) c_{uv} | u \in \Gamma^-(v)\}$
fin si

Théorème 4.3.2 *Soit (C^{LP}, x^{LP}) un point satisfaisant les équations du programme linéaire. Soit S un ordonnancement donné par l'algorithme 4.3.2 et soit ρ' la bore supérieure du ratio donné en définition. Les petits délais de communications locaux assure que $\rho' \leq 1$. Alors, $\forall v \in V$,*

$$C_v \leq \frac{2(1+\rho')}{2+\rho'} C_v^{LP} \leq \frac{4}{3} C_v^{LP}$$

Preuve

Pour une tâche v donnée, soit $\pi(v) = \max\{C_u + (1 - x_{uv}) c_{uv} | u \in \Gamma^-(v)\}$. Ainsi nous pouvons créer une chaîne maximale $\Pi(v) = (\pi_0, \pi_1, \ldots, \pi_l)$ de prédécesseurs avec $\pi_0 = v$ et $\pi_i = \pi^i(v)$. L'élément π_l est minimal dans cet ordre partiel. Après la phase d'arrondis nous avons

$$C_v - C_{\pi(v)} = p_v + (1 - x_{\pi(v)v}) c_{\pi(v)v}$$

Deux cas se présentent à nous :

1. Si $x_{\pi(v)v} = 0$ (après la phase d'arrondis) alors par définition nous avons $C_v - C_{\pi(v)} = p_v + c_{\pi(v)v}$. Avec la phase d'arrondis nous avons $x_{\pi(v)v}^{LP} < 1/2$ et donc

$$C_v^{LP} - C_{\pi(v)}^{LP} \geq p_v + (1 - x_{\pi(v)v}^{LP}) c_{\pi(v)v} > p_v + 1/2 c_{\pi(v)v}$$

Sachant que $\rho' \leq 1$

$$\begin{aligned} \frac{C_v - C_{\pi(v)}}{C_v^{LP} - C_{\pi(v)}^{LP}} &< \frac{p_v + c_{\pi(v)v}}{p_v + 1/2 c_{\pi(v)v}} \\ &= 2 \frac{2p_v + c_{\pi(v)v} - p_v}{2p_v + c_{\pi(v)v}} \\ &= 2 - \frac{2}{2 + \frac{c_{\pi(v)v}}{p_v}} \\ &\leq 2 - \frac{2}{2+\rho'} = \frac{2(1+\rho')}{2+\rho'} \leq \frac{4}{3} \end{aligned}$$

2. Si $x_{\pi(v)v} = 1$ alors $(1 - x_{\pi(v)v}^{LP} \geq 0 = (1 - x_{\pi(v)v})$ et donc

$$C_v - C_{\pi(v)} \leq p_v + (1 - x_{\pi(v)v}^{LP}) c_{\pi(v)v}$$

Dans les deux cas, nous avons

$$C_v - C_{\pi(v)} \leq \frac{2(1+\rho)}{2+\rho} (C_v^{LP} - C_{\pi(v)}^{LP})$$

Soit $\delta := \frac{2(1+\rho')}{2+\rho'}$, et considérons la chaîne $\Pi(v)$ défini précédemment. En sommant la dernière inégalité trouvée sur toutes les tâches faisant partie de ce chemin, nous obtenons :

$$\begin{aligned} C_v - C_{\pi(l)} &= \sum_{i=0}^{l-1}(C_{\pi(i)} - C_{\pi(i+1)}) \\ &\leq \delta \sum_{i=0}^{l-1}(C^{LP}_{\pi_i} - C^{LP}_{\pi_{i+1}}) \\ &= \delta(C^{LP}_v - C^{LP}_{\pi_l}) \end{aligned}$$

Sachant que $\delta \leq 1$ et donc $-\delta C^{LP}_{\pi_l} \leq -C^{LP}_{\pi_l}$ et que par définition nous avons $C_{\pi_l} \geq C^{LP}_{\pi_l}$ alors nous arrivons à

$$C_v - C_{\pi_l} \leq \delta C^{LP}_v - C^{LP}_{\pi_l}$$

et donc $C_v \leq \delta C^{LP}_v$.

□

4.4 Conclusion

Dans ce chapitre nous avons plusieurs résultats. Nous avons proposé un algorithme polynomial de complexité quadratique dans le cas où la duplication est autorisée, et des résultats d'approximation sans duplication.

CHAPITRE

5 Le modèle LCT

Sommaire

5.1 Introduction 95
5.2 Le problème avec duplication 96
5.2.1 Complexité 96
5.2.2 Approximation 99
5.3 Le problème sans duplication 101
5.3.1 Le problème avec m processeurs 101
5.3.2 Approximation 106
5.3.3 Le problème avec une infinité de processeurs 106
5.4 Non-approximability results 107
5.4.1 The minimization of length of the schedule 108
5.4.2 The special case $c = 2$ 112
5.5 A polynomial time for $C_{max} = c + 2$ with $c \in \{2, 3\}$ 114
5.5.1 Approximation 115
5.5.2 Description of the method 115
5.5.3 Analysis of the method 116
5.6 Analysis of our results 117

5.1 Introduction

Nous supposons dans cette partie que le graphe de précédence satisfait la condition suivante :

$$\forall j \in V, \quad \max_{i \in PRED(j)} \{p_i\} \leq \min_{i \in PRED(j)} \{c_{ii}\}$$

Les principaux résultats de ce problème ont été montrés par C.H. Papadimitriou et M. Yannakakis [25] : le problème est $\mathcal{NP}$-complet lorsque les tâches sont unitaires et les délais de communications sont constants et plus grands que un. Les auteurs de l'article ont developpé un algorithme polynômial avec une garantie de performance relative de deux pour le cas général où les durées d'exécution sont arbitraires et les délais de communication d'une tâche générique i vers ses successeurs sont égaux à c_i.

5.2 Le problème avec duplication

5.2.1 Complexité

Théorème 5.2.1 *Le problème de décider si une instance de $\bar{P}|prec; c_{ij} = C > 1; p_i = 1; dup|C_{max}$ pour un graphe $G = (V, E)$ quelconque et avec des entiers τ et C_{max} possède un ordonnancement de longueur au plus C_{max} est $\mathcal{NP}$-complet.*

Preuve

Il est clair que notre problème est $\mathcal{NP}$.

La preuve est basée sur la réduction du problème de la clique au problème $\bar{P}|prec; c_{ij} = C > 1; p_i = 1; dup|C_{max}$ (noté Π).

Soit (G, k) où $G = (V, E)$ est un graphe orienté avec des contraintes de précedences et k un entier naturel. On suppose que G possède au moins $\binom{k}{2}$ arêtes afin qu'il existe dans G une clique de taille k.

Nous allons construire un graphe $D = (U, A)$ et des entiers τ et C_{max} avec τ le délai de communication et C_{max} le temps d'exécution de l'ordonnancement pour le problème Π tel que le graphe D peut être ordonnancé au plus en C_{max} si et seulement si G possède une clique de taille k.

On construit D de la manière suivante :

- pour chaque sommet v de V, on construit un chemin $D_v = (d_{v_1}, d_{v_2}, \ldots, d_{v_{n^2}})$ avec tous les arcs $(d_{v_i}, d_{v_{i+1}})$ où $i \in \{1, \ldots, n^2 - 1\}$ et $n = |V|$.
- pour chaque arête $e = [u, v] \in E$, on construit un sommet étiqueté par C_e (C_e est un successeur immédiat de la dernière tâche des chemins D_u et D_v). On appelle S l'ensemble des nœuds ainsi crée.
- soit t le successeur de tous les C_e avec $\forall e \in E$.

On pose pour la suite $\tau = |V|^2(k-1) + \binom{k}{2}$ et $C_{max} = n^2k + |E|$.

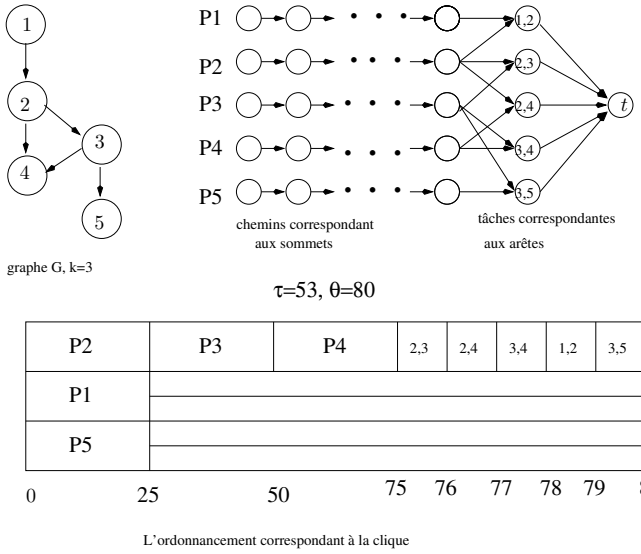


FIG. 5.1 – *Exemple de transformation polynômiale.*

La figure 5.1 donne un exemple où le graphe G contient une clique de taille 3.

Nous allons montrer qu'il existe une clique de taille k dans le graphe G si et seulement si il existe un ordonnancement réalisable pour le problème Π.

- S'il existe une clique de taille k, un ordonnancement réalisable peut-être défini de la manière suivante :
  - on exécute chaque chemin D_v avec v ne faisant pas parti de la clique sur des processeurs différents.
  - on exécute sur le même processeur π les chemins correspondants aux sommets de la clique suivi des tâches représentant les arêtes de la clique.
  - on exécute les arêtes restantes sur π.
  - et pour finir on exécute t sur π.
- On suppose maintenant qu'il existe un ordonnancement réalisable pour le problème Π tel que $C_{max} \leq n^2k + |E|$.

Considérons le sommet t, sa fin d'exécution doit se faire avant C_{max}. L'ensemble de ses prédécesseurs ne peuvent être exécutés que sur le même processeur p du fait des délais de communications.

Nous pouvons les divisés en deux sous-ensembles disjoints S_1 et S_2 :

- soit S_1 l'ensemble des nœuds exécutés avant θ où $\theta = n^2k + \binom{k}{2}$.
- soit S_2 l'ensemble des nœuds exécutés après θ.

On remarque que $C_{max} = n^2k + |E| = \theta + m - \binom{k}{2}$ donc $|S_2| \leq m - \binom{k}{2}$.

Considérons maintenant le nœud $C_{[u,v]} \in S_1$ avec $|S_1| \geq \binom{k}{2}$.

On sait que $C_{[u,v]}$ s'exécute sur p, alors les chemins D_u et D_v ne peuvent s'exécutés que sur le processeur p pour respecter la contrainte de temps (le délai de communication est de $\tau = |V|^2(k-1) + \binom{k}{2}$).

Ceci est vrai $\forall C_{[u,v]} \in S_1$.

En résumé sur le processeur p on exécute le sommet t, tous ses fournisseurs (C_e avec $e \in E$ les nœuds de l'ensemble $\Gamma^-(t)$) et les chemins D_v où v est une extrémité d'une tâche C_e ordonnancé avant θ.

Soit N le nombre de sommets qui sont une extrémité des arêtes correspondant aux nœuds de S_1. Soit T le temps d'ordonnancement sur le processeur p des chemins D_v et des nœuds faisant parti de S_1. On a alors :

$$\begin{aligned} T &\leq \theta \\ Nn^2 + \binom{k}{2} &\leq \theta \end{aligned}$$

On obtient donc $N \leq k$. Sachant qu'il y a au moins $\binom{k}{2}$ arêtes dans S_1 et que G n'admet pas de cycle alors $N = k$ c'est-à-dire que le sous-graphe induit par ces k sommets forme une clique.

On a établi qu'en considérant un graphe $D = (U, A)$ et des entiers τ et C_{max} avec τ le délai de communication et C_{max} le temps d'exécution de l'ordonnancement pour le problème Π que le graphe D peut être ordonnancé au plus en C_{max} si et seulement si G possède une clique de taille k.

En conclusion, le problème de décider si une instance de $\bar{P}|prec; c_{ij} = C > 1; p_i = 1; dup|C_{max}$ pour un graphe $G = (V, E)$ quelconque et avec des entiers τ et C_{max} possède un ordonnancement de longueur au plus C_{max} est $\mathcal{NP}$-complet. □

Corollaire 5.2.1 *Le problème de décider si une instance de $\bar{P}|prec; c_{ij} = c > 1; p_i = 1|C_{max}$ pour un graphe $G = (V, E)$ quelconque et avec des entiers τ et C_{max} possède un ordonnancement de longueur au plus C_{max} est $\mathcal{NP}$-complet.*

Preuve La preuve précédente n'utilise pas la notion de duplication. □

5.2.2 Approximation

Dans cette partie nous allons voir un algorithme 2-approché pour le $\bar{P}|prec; c_{ij} = c > 1; p_i = 1; dup|C_{max}$. Ce qui est remarque dans cet algorithme approché, cést que la borne ne dépend pas de la valeur de c.

Nous allons calculer sur la profondeur du graphe une fonction $e(v)$.

Algorithme 5.1 Algorithme donnant les bornes inférieures $e(v)$ d'exécution d'une tâche v

si $\Gamma^-(v)$ **alors**
$e(v) = 0$
sinon
Soit $\Gamma^-(v) = \{u_1, u_2, \ldots, u_p\}$ l'ensemble des prédécesseurs de v classé de manière décroissante concernant leur valeur e ($e(u_1) \geq e(u_2) \geq \ldots \geq e(u_p)$) c'est à dire nous avons défini une ordre de priorité.
Soit $k = \min\{c + 1, p\}$
Nous définissons $e(v) = e(u_k) + k$
fin si

Algorithme 5.2 Algorithme donnant un orodnnancement réalisable

Nous ordonnançons les tâches v à l'instant $2e(v)$. Le processeur exécutant la tâche v exécutent également les c tâches prédécesseurs de v ayant la plus grande priorité, et la tâche v recevra le reste des informatiosn nécessaire à son exécution par les communications.

Maintenant nous allons montrer que l'algorithme 5.2.2 donnent bien les bornes inférieures de début d'exécution et l'algorithme 5.2.2 conduit à un ordonnancement valide.

Lemme 5.2.1 *Il n'existe pas d'ordonnancement pour lequel la tâche v s'exécute avant l'instant $e(v)$. Ainsi, $e(v)$ est une borne inférieure de début d'exécution.*

Preuve

Avant toute chose, il est important de noter le fait que de déterminer la borne inférieereure ne conduit à l'existence d'un ordonnancement réalisable.

Nous allons procéder par induction sur la profondeur du graphe.

- Le résultat est vrai pour une source v.
- Supposons par hypothèse de récurrence que v peut-être exécutée à $t < e(v)$, et considérons les k prédécesseurs $u_1, \ldots, u_k$ de v admettant la plus grande valeur de e (regarder la définition de k dans ce contexte).

Par hypothèse de récurrence, chaque tâche u_i est exécutée à ou après l'instant $e(u_i) \geq e(u_k)$. Sachant que $t < e(v) = e(u_k) + k \leq e(u_k) + c + 1$, alors chacun des sommets u_i est ordonnancé avant $c + 1$ instant avant v, et ainsi sur le même processeur que v. Nous savons qu'il existe k tâches exécutées entre les instants $e(u_k)$ et t, nous avons $t \geq e(u_k) + k$, or $t \geq e(v)$, en contradiction avec notre hypothèse.

□

Lemme 5.2.2 *Pour n'importe quelle tâche v, il existe une ordonnancement valide pour lequel la tâche v est exécutée à l'instant $2e(v)$*

Preuve

Nous allons également procéder par récurrence :

- Le résultat est vrai pour une source v.
- Par hypothèse de récurrence, supposns en premier lieu que le sommet v admet $p \leq c + 1$ prédécesseurs. Alors nous pouvons les exécuter sur le même processeur que v jusqu'à $e(v) + 1 \leq 2e(v)$.
  Supposons maintenant que v admet $p > c + 1$ prédécesseurs. Nous pouvons les ordonner par valeur e décroissante : $e(u_1) \geq e(u_2) \geq \ldots \geq e(u_p)$. Sachant que $e(v) = e(u_{c+1}) + c + 1$, nous obtenons $e(v) - c - 1 \geq e(u_i), \forall j \geq c + 1$. Ainsi, par induction, tous les prédécesseurs de v à l'exception des c premières tâches, peuvent être ordonnancées sur des processeurs différents à l'instant $2e(v) - 2c - 2$. Ce qui signifie que en commençant à l'instant $2e(v) - 2c - 1$, les c prmeières tâches, nous obtenons un ordonnancement valide, et v peut s'exécutée à l'instant $2e(v)$.

□

Théorème 5.2.2 *L'algorithme 5.2.2 admet une performance relative de deux pour le $\bar{P}|prec; c_{ij} = c > 1; p_i = 1; dup|C_{max}$.*

Preuve

Avec les Lemmes 5.2.2 et 5.2.1, nous pouvons conclure ; □

Corollaire 5.2.2 *L'algorithme 5.2.2 est également valide pour le problème $\bar{P}|prec; c_{ij} > 1; p_i = 1; dup|C_{max}$ et sa performance relative est de 2.*

Preuve

La preuve est identique. A vous de la faire

□

5.3 Le problème sans duplication

5.3.1 Le problème avec m processeurs

5.3.1.1 Complexité

5.3.1.2 Algorithme polynomial

Dans cette partie, nous allons nous intéressé au problème $P|prec, c_{ij} = c, p_i = 1|C_{max} \leq c+1$. IL est clair que toute paire de tâches relié par un arc doivent séxécutées sur le même processeur. Alors, nous pouvons décider si un graphe de précédence est exécutable en $C_{max} \leq c+1$ en utilisant la procédure suivante :

Algorithme 5.3 Algorithme donnant la solution optimale

Déterminer les composantes connexes du graphe de précédence $B_1,\ B_2, \ldots, B_k$.
Soit $|B_i|$ le nombre de tâches de B_i. Si $\forall i,\ 1 \leq i \leq k,\ |B_i| \leq C_{max}$ alors il suffit de résoudre le problème d'ordonnancement multiprocesseurs pour des tâches indépendantes $T_1,\ T_2, \ldots, T_k$ avec des durées déxécution $|B_1|,\ \ldots, |B_k|$ respectivement. Sachant que $|B_i| \leq C_{max} \leq c+1, \forall i$ ceci peut-ëtre fait en un temps polynomiale . Dans le cas contraire, le graphe de précédence ne peut être exécutée en C_{max}.

5.3.1.3 Np-complétude

Maintenant nous allons montrer que $C_{max} = c+3$ le problème est $\mathcal{NP}$-complet.

Théorème 5.3.1 *Le problème de décider si une instance de $P|prec, c_{ij} = c, p_i = 1|C_{max}$ admet un ordonnancement de longueur au plus $c+3$ est $\mathcal{NP}$-complet.*

Preuve

Nous allons basé notre réduction sur le problème BBCG, définie précédemment. Nous allons construire une instance du problème d'ordonnancement de la manière suivante :

- Nous avons $m = n+2$ processeurs où n correspond au nombre de sommets du graphe $B = (X, Y)$ pour exécutées $n+2)(c+3)$ tâches.
- Le graphe de précédence est le suivant :

1. Deux chemins $A_1 = a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_{c+3}$ et $A_2 = b_1 \rightarrow b_2 \rightarrow \ldots \rightarrow b_{c+3}$
2. $B = X \cup Y$ où $X = \{x_1, \ldots, x_n\}$ et $Y = \{y_1, \ldots, y_n\}$ et les arêtes entre X et Y sont remplacées les arcs correspondant orienté de X vers Y.
3. $\cup_{i=1}^{c+1} U_i$ où $U_i = \{u_i^1, u_i^2, \ldots, u_i^{n/2}\}$ avec $u_i^j \rightarrow u_{i+1}^j, \forall 1 \leq i \leq c$ et $1 \leq j \leq n/2$.
4. $\cup_{i=1}^{c+1} V_i$ où $V_i = \{v_i^1, v_i^2, \ldots, v_i^{n/2}\}$ avec $v_i^j \rightarrow v_{i+1}^j, \forall 1 \leq i \leq c$ et $1 \leq j \leq n/2$.
5. Les tâches a_{c+3} et b_{c+3} sont précédées par tous les tâches de X et de U_2
6. Les tâches a_1 et b_1 précédent toutes les tâches Y et V_c.

- Supposons que le graphe B admettent un sous ensemble indépendent équilibré (X_1, Y_1) avec $X_1 \subset X$, et $Y_1 \subset Y$, et $|X_1| = |Y_1| = n/2$, alors montrons qu'il existe un ordonnancement de longueur $c+3$.
  Construisons cet ordonnancement :
  - A $t = 0$ nous exécutons les tâches $a_1,\ b_1, X - X_1$ et U_1.
  - A $t = 1$ nous exécutons les tâches $a_2,\ b_2, X_1$ et U_2.
  - A $t = i$ avec $2 \leq i \leq c$, nous exécutons les tâches $a_i,\ b_i,\ V_{i-2}$ et U_i.
  - A $t = c+1$, nous exécutons les tâches $a_{c+2},\ b_{c+2},\ V_c$ et Y_1.
  - A $t = c+3$, nous exécutons les tâches $a_{c+3},\ b_{c+'3},\ V_{c+1}$ et $Y - Y_1$

  Il est facile de voir, que l'ordonnancement est réalisable avec un placement adéquate des tâches sur les processeurs. De manière plus précise, nous exécutons consécutivement sur le même processeur toutes les tâches du chemins A_1 (resp. A_2). Soit U^j l'ensemble des tâches $u_i^j,\ 1 \leq i \leq c+1$ avec un j fixé. Alors toutes les tâches de U^j sont exécutées sur le même processeur. De manière similaire, soit V^j l'ensemble des tâches $v_i^j,\ 1 \leq i \leq c+1$ et pour j fixé. Toutes les tâches de V^j sont assignées sur le même processeur. alors, la faisabilité de l'ordonnancement est garantit par l'existence d'un sous-ensemble indépendent équilibré (X_1, Y_1).
- Supposons qu'il existe un ordonnancement de longueur $c+3$, montrons alors l'existence d'un sous ensemble indépendent équilibré (X_1', Y_1') avec $X_1' \subset X$, et $Y_1' \subset Y$, et $|X_1'| = |Y_1'| = n/2$ dans le graphe B.
  Nous allons faire quelques remarques essentielles :
  - Sachant que le nombre de tâches est exactement $(n+2)(c+3)$ et la longueur de l'ordonnancement est de $c+3$ alors il n'existe pas de temps d'inactivité.

– Pour chaque chemin de longueur $c+3$, il est clair que les tâches du chemin doivent s'exécutées consécutivement sur un processeur. Ainsi, deux processeurs servent à exécutées les deux chemins.
– Sachant que les tâches a_{c+3} et b_{c+3} sont précédées par les tâches de l'ensemble X, et dans le but de préserver le deadline, les tâches de l'ensemble X doivent s'exécutées sur les deux proemiers instants. De plus, sachant que les tâches a_{c+3} et b_{c+3} sont précédées par les tâches de U_2 et qu'il existe un couplage entre les tâches de U_1 et de U_2 ; alors les tâches de U_1 et U_2 sont ordonnancées durant les deux premiers instants, et plus précisément, toutes les tâches de U_1 doivent être exécutées $t=0$ et les tâches de U_2 à $t=1$ sur le même processeur à cause du couplage. Ainsi, il y a $n/2+2$ processeurs qui exécutent les tâches $a_1,\ b_1$ et U_1 (resp. a_2, b_2 et U_2 à $t=0$ (resp. $t=1$).
– Dans le but de respecter les contraintes, toutes les tâche sde X doivent être exécutées durant les deux premiers instants ; nous devons occupé tous les processeurs ainsi l'ensemble X est scindé en deux sous-ensemble X'_1 et X'_2 tel que $|X'_1| = |X'_2| = n/2$. Sans perte de généralité, nous supposons que X'_1 est exécuté avant X'_2. De plus, nous pouvons voir X'_1 et X'_2 sont ordonnancées sur les même processeurs.
– Sachant que toutes les tâches de Y sont précédées par a_1 et b_1, la borne inférieure de début d'exécution pour les tâches de Y est $(c+2)$. De manière similaire, les tâches de V_c sont précédées par a_1 et b_1, aucune tâche de V_c ne peut commencer son exécution avant $(c+2)$ et ainsi, V_c et V_{c+1} son ordonanncées de manière consécutive occupant de cette manière $n/2$ processeurs durant l'intervalle $[(c+2),(c+3)]$. Ceci signifie que $n/2+2$ processeurs sont nécessaires pour l'exécution de $a_{c+2},\ b_{c+2}$ et V_c (resp. $a_{c+3},\ b_{c+3}$ et V_{c+1}). Ainsi, dans le but de respecter les contraintes, toutes les tâches de Y doivent être exécutées durant les deux derniers instants, tous les processeurs doivent être ocuupés alors l'ensemble Y sera scindé en deux sous ensemble Y'_1 et Y'_2 tel que $|Y'_1| = |Y'_2| = n/2$. xans pert e de généralité, nous assumons que les tâches de Y'_1 (resp. Y'_2) seront exécutées à l'instant $t=c+1$ (resp. $t=c+2$).
– Durant les deux derniers instants de l'ordonnancement tous les processeurs sont actifs, nous devons exécutées toutes les tâches de U^j en $c+1$ intervalle de temps correspondant à la longueur de ces chemins. Donc, il est facile de voir avec $c \geq 2$ que toutes les tâches d eU^j sont ordonnancées sur le même processeur et de manière consécutive. Ceci est également valable pour chaque chemins V^j (dàa ce stade de la démonstration, tous les processeurs sont actifs durant les deux premiers instants de l'ordonnancement). Sachant que deux chemins de longueur $(c+1)$ ne peuvent

s'exécutés sur le même processeur, nous avons que l'ensemble des processeurs, noté par la suite $\mathcal{P}_U$, exécutant les tâches de U_i est distinct, noté par la suite $\mathcal{P}_V$ de l'ensemble des processeurs exécutant les tâches V_j. Ceci, implique que les tâches de X sont exécutées sur $\mathcal{P}_v$ et les tâches de Y sur $\mathcal{P}_U$.

Alors, les tâches de X'_2 sont ordonnancées à l'instant deux et les tâches de Y'_1 à l'instant $(c+1)$ surdes processeurs différents processeurs. Il n'y a pas de temps de communiquer alors $X'_2 \cup Y'_1$ forment un ensemble indépendent équilibré.

□

Corollaire 5.3.1 *Le problème de décider si une instance de $P|prec, c_{ij} = c, p_i = 1, dup|C_{max}$ admet un ordonnancement de longueur au plus $c+3$ est $\mathcal{NP}$-complet.*

Preuve

La preuve vient directement de celle du Théorème 5.3.1. En effet, aucune tâche ne peut être dupliquée sinon le nombre de tâches seraient plus grand que $(c+3)(n+2)$ et ainsi l'ordonnancement serait plus grand que $(c+3)$.

□

Maintenant nous allons étendre le résultat précédent de $\mathcal{NP}$-completude pour les graphes bipartis.

Théorème 5.3.2 *Le problème de décider si une instance de $P|biparti, c_{ij} = c, p_i = 1|C_{max}$ admet un ordonnancement de longueur au plus $c+3$ est $\mathcal{NP}$-complet.*

Preuve

Nous allons nous basé sur le problème $\mathcal{NP}$-complet Clique. Soit un graphe non orienté $G=(V,E)$ et un entier k, nous allons construire une instance de $P|biparti, c_{ij} = c, p_i = 1|C_{max}$ de la manière suivante :

– Soit $l = \frac{k(k+1)}{2}$ le nombre d'arêtes de la clique de taille k. Nous définissons le nombre de processeurs m comme étant égal à $m = 2(m'+1)$ avec $m' = \max\{|V|+l-k, |E|-l\}$.
– Chaque sommet $v \in V$ correspond à uns étoile faîtes des tâches-sommets $J_v,\ K_v^1, \ldots, K_v^c$ et les contriantes de précédence entre ses tâches sont les suivantes : $J_v \rightarrow K_v^1, \ldots, J_v \rightarrow K_v^c$.
– Chaque arête $[u,u'] \in E$ correspond une tâche-sommet $L_{[u,u']}$ précédée par les tâches-sommets J_u et $J_{u'}$, c'est à dire $J_u \rightarrow L_{[u,u']}$ et $J_{u'} \rightarrow L_{[u,u']}$.
– Nous considérons maintenant quatre ensembles de tâches distinctes X ,Y, U et W de cardinalité respective $|X| = m-l-|V|+k$, $|Y| =$

$m - |E| + l$, $|U| = m - k$ et $|W| = m - |V|$. La construction est complété par une graphe complet entre tous les sommets de $U \rightarrow X$, $U \rightarrow Y$ et $W \rightarrow Y$.

Notation Nous notons par $\mathcal{C}$ les sommets élément de la clique de taille k.

Clairement le graphe de précédence obtenu est un graphe biparti.

- Supposons que le graphe G contienne une clique de taille k, nous allons montrer comment nous pouvons ordonnancer en $(c+3)$ unités de temps.
  Construisosn l'ordonnancement :
  - A $t = 0$, nous exécutons les tâches de l'ensemble U et les tâches $\mathcal{J}_{\mathcal{C}} = \{J_v | v \in \mathcal{C}\}$.
  - A $t = 1$ nous exécutons les tâches de l'ensemble W et les tâches de $\mathcal{K_C}^1 = \{K_v^1 | v \in \mathcal{C}\}$, $\mathcal{J}_{V-\mathcal{C}} = \{J_v | v \in V - \mathcal{C}\}$.
  - A $t = i$ avec $3 \leq i \leq c+1$, nous exécutons les tâches $\mathcal{K_C}^{i-1} = \{K_v^{i-1} | v \in \mathcal{C}\}$ et $\mathcal{K_C}^{i-2} = \{K_v^{i-2} | v \in V - \mathcal{C}\}$.
  - A $t = c+1$, nous exécutons les tâches de l'ensemble X et les tâches de $\mathcal{K}^c_{V-\mathcal{C}}$, $\mathcal{L}_{\mathcal{C}} = \{L_{[u,u']} | v,\ u \in \mathcal{C}\}$.
  - A $t = c+2$, nous exécutons les tâches de l'ensemble Y et les tâche sd e$\mathcal{L}_{V-\mathcal{C}}$.

  Il st facile de voir que chaque tâche K_v^i est exécutée sur le même processeur que J_v, alors le nombres de processeurs est vérifié à chaque instant de l'ordonnancement.
- Supposons qu'il existe un ordonnancement en $(c+3)$ unités de temps, nous allons qu'il existe une clique de taille k dans le graphe G.
  Nous allons fair equelques remarques essentielles sur l'ordonnancement :
  - Aux vues du nombre de tâches , il est clair que tous les processeurs sont tous actifs à chaque instant.
  - L'ensemble U est un ensemble dominant alors toutes les tâches de cet ensemble devront être exécutées à $t = 0$. En effet, dans le cas contraire $|X| + |Y| - 1$ tâches ou plus, seraient exécutées à l'instant $t = c+2$, l'égalité est atteinte si une seule tâche de U est exécutée à l'instant $t = 1$. Mais nous avons $|X| + |Y| - 1 > m$, impossible.
  - Sachant $|U| = m - k$, l'ensemble des tâches du type $\mathcal{J}$ peuvent être ordonnancées à l'instant $t = 0$, appelons par la suite par $\mathcal{J}_{\mathcal{A}}$ cet ensemble. Nous avons $\mathcal{J}_{\mathcal{A}} \leq k$. Ainsi, au plus l tâches du type $\mathcal{L}$ peuvent être exécutées à l'instant $t = c+1$ et au plus $|E| - l$ tâches de type $\mathcal{L}$ doivent être ordonnancées à l'instant $t = c+2$.
  - Sachant qu'aucune tâche de l'ensemble Y ne peut être exécutée avant l'instant $t = c+2$ (car $|U| + |W| > m$ et quelques tâches de W doivent être exécutées à l'instant $t = 1$), nous avons nécessairement au moins $|Y| + |E| - l = m$ processeurs çà l'instant $t = c+2$.

Dans le but d'avoir un ordoanncement réalisable, il est nécessaire que $|\mathcal{J}_{\mathcal{A}}| = k$ et au moins l tâches du type $\mathcal{L}$ exécutées à l'instant $t = c+1$. Ceci est possible si le graphe G contient une clique de taille k avec $\mathcal{J}_{\mathcal{A}}$ comme ensemble des sommet de la clique.

□

Corollaire 5.3.2 *Le problème de décider si une instance de $P|biparti, c_{ij} = c, p_i = 1, dup|C_{max}$ admet un ordonnancement de longueur au plus $c+3$ est $\mathcal{NP}$-complet.*

Preuve La preuve vient directement de celle du Théorème 5.3.2. En effet, aucune tâche ne peut être dupliquée sinon le nombre de tâches seraient plus grand que $(c+3)(n+2)$ et ainsi l'ordonnancement serait plus grand que $(c+3)$.

□

Corollaire 5.3.3 *Il n'existe pas d'algorithme approché avec une performance relative inférieure à $1 + \frac{1}{c+3}$ pour le problème $P|biparti, c_{ij} = c, p_j = 1|C - max$ sous l'hypothèse que $\mathcal{NP} \neq \mathcal{P}$.*

5.3.2 Approximation

5.3.3 Le problème avec une infinité de processeurs

5.3.3.1 Résultats précédents

Contrary to the complexity results, as we know, a unique approximation algorithm is given by Rapine [27]. The author gives the lower bound $O(c)$ for the list scheduling in the presence of large communication delays. Approximation results are given in Table 5.1. Notice that, as known, there is no approximation algorithm with ratio guarantee better than the trivial bound : $(c+1)$.

5.3.3.2 Complexité

The problem Π_1 is a variant of the well known SAT problem [10]. We will call this variant the $One\text{-}in\text{-}(2,3)SAT(2,\bar{1})$ problem that we will denote as Π_2. Let n be a multiple of 3 and let $\mathcal{C}$ be a set of clauses of cardinality 2 or 3. There are n clauses of cardinality 2 and $n/3$ clauses of cardinality 3 so that :

- each clause of cardinality 2 is equal to $(x \vee \bar{y})$ for some $x,\ y \in \mathcal{V}$ with $x \neq y$.
- each of the n literals x (resp. of the literals $\bar{x}$) for $x \in \mathcal{V}$ belongs to one of the n clauses of cardinality 2, thus to only one of them.

Machines	c_{ij}	Approximation	References
$\bar{P}$	$c = 1$	$\rho \leq 4/3$	[21]
P	$c = 1$	$\rho \leq 7/3$	[20]
$\bar{P}$	$c \geq 2$	2 for tree	[19]
$\bar{P}$	$c \geq 2$	?	?
P	$c \geq 2$	$O(c)$	[27]

TAB. 5.1 – Approximation results

– each of the n literals x belongs to one of the $n/3$ clauses of cardinality 3, thus to only one of them.
– whenever $(x \vee \bar{y})$ is a clause of cardinality 2 for some $x,\ y \in \mathcal{V}$, then x and y belong to different clauses of cardinality 3.

Question :

Is there a truth assignment for $I : \mathcal{V} \rightarrow \{0, 1\}$ such that every clause in $\mathcal{C}$ has exactly a true literal ?

In order to illustrate Π_1, we consider the following example.

Example The following logic formula is a valid instance of Π_2 :

$(x_0 \vee x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5) \wedge (\bar{x}_0 \vee x_3) \wedge (\bar{x}_3 \vee x_0) \wedge (\bar{x}_4 \vee x_2) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_5 \vee x_1) \wedge (\bar{x}_2 \vee x_5)$.

The answer to Π_1 is yes. It suffices to choose $x_0 = 1$, $x_3 = 1$ and $x_i = 0$ for $i = \{1, 2, 4, 5\}$. This yields a truth assignment satisfying the formula, and there is exactly one true literal in every clause. For the proof of the $\mathcal{NP}$-completeness see [11].

5.4 Non-approximability results

In this section we show in the first part, the problem denoting by $\bar{P}|prec; c_{ij} = c \geq 3; p_i = 1|C_{max}$ does not possess a polynomial time approximation algorithm with ration guarantee better than $1 + \frac{1}{c+4}$ for the minimization of the length of the schedule (resp. $1 + \frac{1}{2c+5}$, for the minimization of the sum of the completion time). We also give the $\mathcal{NP}$-completeness for the special case $c = 2$.

5.4.1 The minimization of length of the schedule

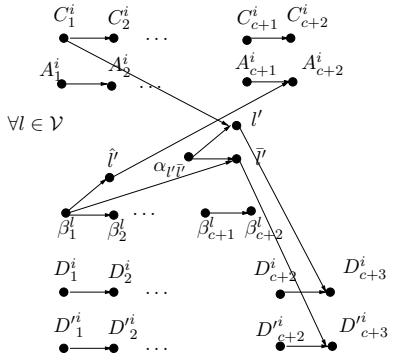


FIG. 5.2 – *A partial precedence graph*

Théorème 5.4.1 *The problem of deciding whether an instance of* $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max}$ *has a schedule of length at most* $(c + 4)$ *is* $\mathcal{NP}$*-complete with* $c \geq 3$.

Preuve It is easy to see that $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max} = c + 4 \in \mathcal{NP}$.

Our preuve is based on a reduction from Π_1.

Given an instance π^* of Π_1, we construct an instance π of the problem $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max} = c + 4$, in the following way :

1. For all $x \in \mathcal{V}$, we introduce $(c + 6)$ variables-tasks : $\alpha_{x'\bar{x}'}$, x', $\bar{x}'$, $\hat{x}'$, β^x_j with $j \in \{1, 2, \ldots, c + 2\}$. We add the precedence constraints : $\alpha_{x'\bar{x}'} \rightarrow x'$, $\alpha_{x'\bar{x}'} \rightarrow \bar{x}'$, $\beta^x_1 \rightarrow \hat{x}'$, $\beta^x_1 \rightarrow \bar{x}'$, $\beta^x_j \rightarrow \beta^x_{j+1}$ with $j \in \{1, 2, \ldots, c + 1\}$.
2. For all clauses of length three denoted by $C_i = (y \vee z \vee t)$, we introduce $2 \times (2 + c)$ clauses-tasks C^i_j and A^i_j, $j \in \{1, 2, \ldots c + 2\}$, with precedence constraints : $C^i_j \rightarrow C^i_{j+1}$ and $A^i_j \rightarrow A^i_{j+1}$, $j \in \{1, 2, \ldots, c+1\}$. We add the constraints $C^i_1 \rightarrow l$ with $l \in \{\hat{y}', z', t'\}$ and $l \rightarrow A^i_{c+2}$ with $l \in \{\hat{y}', \hat{z}', \hat{t}'\}$.
3. For all clauses of length two denoted by $C_i = (x \vee \bar{y})$, we introduce $(c + 3)$ clauses-tasks D^i_j, $j \in \{1, 2, \ldots, c + 3\}$ with precedence constraints : $D^i_j \rightarrow D^i_{j+1}$ with $j \in \{1, 2, \ldots, c + 2\}$ and $l \rightarrow D^i_{c+3}$ with $l \in \{x', \bar{y}'\}$.

The above construction is illustrated in Figure 5.2. This transformation can be clearly computed in polynomial time.

- Let us first assume that there is a schedule of length at most $(c+4)$. In the following, we will prove that there is a truth assignment $I : \mathcal{V} \rightarrow \{0,1\}$ such that each clause in $\mathcal{C}$ has exactly one true literal.
  First we can remark that if $c \geq 3$ then $2c+2 > c+4$ and so, each path $A^i_j,\ \beta^x_j,\ C^i_j$ or $D^i_{j'}$ with $j \in \{1,2,\ldots,c+2\}$ and $j' \in \{1,2,\ldots,c+3\}$ must be executed on the same processor. In more, two of these paths can not be executed on the same processor.
  Notation : In the following we denote by P_A (resp. P_C) the set of the $\frac{n}{3}$ processors which execute a path A^i_j (resp. a path C^i_j). Notice that we know by the definition of the problem Π_1, that in an instance admits $\frac{n}{3}$ clauses of length three where n denotes the number of variables. In the same way, we denote by P_β (resp. P_D) the set of the n processors witch execute a path β^x_j (resp. a path D^i_j).

Lemme 5.4.1 *If $C_{max} = c+4$ then assigning the value true to the variable x if and only if the variable-task x' is executed on a processor of the path P_C we obtain a correct solution.*

Preuve $\forall l \in \mathcal{V}$, by construction, it is clear that :
- Each variable-task l' is executed on a processor de P_C at slot 3 or on a processor of P_D at slot $(c+2)$ or $(c+3)$,
- Each variable-task $\bar{l}'$ is executed on a processor de P_β at slot 3 or on a processor of P_D at slot $(c+2)$ or $(c+3)$,
- Each variable-task $\hat{l}'$ is executed on a processor de P_β at slot 3 or on a processor of P_A at slot $(c+2)$ or $(c+3)$,
- The variables-tasks $\bar{l}'$ and $\hat{l}'$ can not be executed together on a processor of P_β (they have a common predecessor).

Notation and property : For each $l \in \mathcal{V}$, we can associate the three tasks l', $\bar{l}'$, $\hat{l}'$. We denote by $X = \{l' | l \in \mathcal{V}\}$, $\bar{X} = \{\bar{l}' | l \in \mathcal{V}\}$ and $\hat{X} = \{\hat{l}' | l \in \mathcal{V}\}$ three sets of tasks. At each subset A of $\bar{X}$ (resp. $\hat{X}$), we can associate a subset B of X in the following way : $l' \in B$ *if and only if* $\bar{l}' \in A$ (resp. $\hat{l}' \in A$).
Let be :
- X_1 is the set containing the variable-task l' such that variable-task l' is executed on a processor of P_C,
- X_2 is the set containing the variable-task l' such that the variable-taskl' is executed on a processor of P_D,
- X_3 is the set containing the variable-task l' such that the variable-task $\bar{l}'$ is executed on a processor of P_β,

- X_4 is the set containing the variable-task l' such that the variable-task$\bar{l}'$ is executed on a processor of P_D,
- X_5 is the set containing the variable-task l' such that the variable-task $\hat{l}'$ is executed on a processor of P_β,
- X_6 is the set containing the variable-task l' such that the variable-task $\hat{l}'$ is executed on a processor of P_A.

Let be $x_i = |X_i|$ for $i \in \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$.
We can stem from the construction of an instance of the scheduling problem,

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

So we obtain :

$$x_1 + x_2 = n \tag{5.1}$$
$$x_3 + x_4 = n \tag{5.2}$$
$$x_5 + x_6 = n \tag{5.3}$$
$$x_1 \leq \frac{n}{3} \tag{5.4}$$
$$x_6 \leq \frac{2n}{3} \tag{5.5}$$
$$x_3 + x_5 \leq n \tag{5.6}$$
$$x_2 + x_4 \leq n \tag{5.7}$$

Indeed,
- $(1),\ (2),\ (3)$ We must execute all the tasks of the sets X, $\bar{X}$ and $\hat{X}$.
- (4) On the processor which execute the path C^i_j of the clause $C_i = (y \vee z \vee t)$, we can execute at most one of the three variables-tasks y', z', t'. Indeed, all variables-tasks l' as a successor which is executed on a processor of P_D. If it is executed on the proceesor which scheduled the tasks from the path P_C it does not be executed before the slot 3 and so, the variable-task $\alpha_{l'\bar{l}'}$ must be executed on the same processor which becomes satured. So, we have $|X_3| < |P_C|$.
- (5) Each processor of the paths P_A has two free slots and $|P_A| = \frac{n}{3}$.
- (6) All the variables-tasks $\bar{l}'$ or $\hat{l}'$ which are executed on a processor of the path P_β must be finished before slot 3 (it has a successor executed on another processor). So the variable-task $\alpha_{l'\bar{l}'}$ must be executed on the same processor which becomes satured. Therefore, at most one task between the variables-tasks $\bar{l}'$ and $\hat{l}'$ can be executed on a processor of the path P_β and so, $|X_3| + |X_5| \leq |P_\beta|$.

– (7) It is cleat that, $|P_D| = n$ and there is at most one free slot on each processor of P_D.
As $x_3 + x_5 = n$ and as $\forall l'$ only one variable-task between the variables-tasks $\bar{l}'$ and $\hat{l}'$ can be executed on a processor of P_β, we have $X_3 \cap X_5 = \emptyset$. Consequently, we obtain $X_3 \cup X_5 = X$. As the set X_4 (resp. X_6) is the complementary of the set X_3 (resp. X_5) we have $X_4 \cup X_6 = X$. In more if the variable-task l' is executed on a processor of P_C then the variable-task $\alpha_{l'\bar{l}'}$ is executed on the same processor. The variable-task $\bar{x}'$ can not be executed before the slot $(c+2)$ therefore on a processor of P_D. We can deduce that $X_1 = X_4$ (the two set are the same cardinality). Finally we have $X_1 \cup X_2 = X$, $X_3 \cup X_4 = X$, $X_5 \cup X_6 = X$, $X_4 \cup X_6 = X$, $X_3 \cup X_5 = X$, $X_1 = X_4$ and therefore $X_1 = X_4 = X_5$ and $X_2 = X_3 = X_5$.

We can deduce by the prevouisly equations that $x_1 = x_4 = x_5 = \frac{n}{3}$ and $x_2 = x_3 = x_6 = \frac{2n}{3}$.

□

So if we affect the value "true" to the variable l if and only the variable-task l' is executed on a processor of P_C is trivial to see that in the clause of length 3 we have one and only one literal equal to "true".
Let be $c = (x \vee \bar{y})$, a clause of length 2.
– If $x' \in X_1 \implies y' \in X_4 \implies y' \in X_1$. The first implication is due to the fact that each processor of the path P_D must be satured ($x_2 + x_4 = n$). The second, because $X_1 = X_4$. Only the literal x is "true" between the variables x and $\bar{y}$.
– If $x' \in X_2 \implies y' \in X_3 \implies y' \in X_2$. The first implication is due to the fact that there is only one free slot on each processor executing the path P_D. The second, because $X_3 = X_2$. Only the literal $\bar{y}$ is "true" between the variables x and $\bar{y}$

- Conversely, we suppose that there is a truth assignment $I : \mathcal{V} \to \{0, 1\}$, such that each clause in $\mathcal{C}$ has exactly one true literal.
Suppose that the true litteral in the clause $C_i = (y \vee z \vee t)$ is t. Therefore, the variable-task t' (resp. y' and z') is processed at the slot 3 (resp. at the slot $(c+2)$) on the same processor as the path P_{C_i} (resp. as the path P_D and $P_{D'}$, where D and D' design a clause of length two where the variables y and z occured). The $\frac{2n}{3}$ other variables-tasks y' not yet schedule are executed at slot 3 on processor P_β as the variable-task $\alpha_{y'\bar{y}'}$. The variable-task $\hat{t}'$ (resp. $\hat{y}'$ and $\hat{z}'$) are executed at the slot 2 (resp. $c+2$ and $c+3$) on a processor of the path P_β (resp. P_A).
This concludes the preuve of Theoreme 5.4.1.

□

5.4.2 The special case $c = 2$

In this part we will prove that the problem $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max}$ has a schedule of length at most six is $\mathcal{NP}$-complete.

Théorème 5.4.2 *The problem of deciding whether an instance of $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max}$ has a schedule of length at most six is $\mathcal{NP}$-complete.*

Preuve It is easy to see that $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max} = 6 \in \mathcal{NP}$.
Our preuve is based on a reduction from Π_1.
Given an instance π^* of Π_1, we construct an instance π of the problem $\bar{P}|prec; c_{ij} = 2; p_i = 1|C_{max} = 6$, in the following way :

1. For all $x \in \mathcal{V}$, we introduce 5 variables-tasks : $\alpha_{x'}$, x', $\bar{x}'$, $\hat{x}'$, $\beta_{x'}$. We add the precedence constraints : $\alpha_{x'} \to x'$, $\alpha_{x'} \to \bar{x}'$, $\beta_{x'} \to \hat{x}'$, $\beta_{x'} \to \bar{x}'$.
2. For all clauses of length three denoted by $C_i = (y \vee z \vee t)$, we introduce two clauses-tasks C^i and A^i. We add the following precedence constraints : $C^i \to l$ with $l \in \{y', z', t'\}$ and $l \to A^i$ with $l \in \{\hat{y}', \hat{z}', \hat{t}'\}$.
3. For all clauses of length two denoted by $C_i = (x \vee \bar{y})$, we introduce five clauses-tasks D^i_j, $j \in \{1, 2, \ldots, 5\}$ with precedence constraints : $D^i_j \to D^i_{j+1}$ with $j \in \{1, 2, 3, 4\}$ and $l \to D^i_5$ with $l \in \{x', \bar{y}'\}$.

- Let us first assume that there is a schedule of length at most 6. In the following, we will prove that there is a truth assignment $I : \mathcal{V} \to \{0, 1\}$ such that each clause in $\mathcal{C}$ has exactly one true literal.
First we can remark that if a task is executed at slot 3 (resp. at slot 4) or before (resp. after) all these predecessors (resp successors) are executed on the same processor. Since the communication is allowed on a path of length 5 so $\forall i$ all the clauses-tasks D^i_j are executed on the same processor.

Lemme 5.4.2 *If $C_{max} = 6$ then assigning the value true to the variable x if and only if the variable-task x' is executed at slot 3 we obtain a correct solution.*

Preuve $\forall l \in \mathcal{V}$, by construction, it is clear that :
– The variable-task l' is executed at slot 3 on the processor which executed $\alpha_{x'}$ and C^i where $x \in C_i$ or after slot 4 on the processor which executed D^i_5 such that $x \in C_i$.

– The variable-task $\bar{l}'$ is executed at slot 3 on the processor which executed $\alpha_{x'}$ and $\beta_{x'}$ or after slot 4 on the processor which executed D_5^i such that $\bar{x} \in C_i$.
– The variable-task $\hat{l}'$ is executed at slot 2 or 3 on the processor which executed $\beta_{x'}$ or after slot 4 on the processor which executed A^i such that $x \in C_i$.

□

Using the same notation as previously, we consider
– X_1 is the set containing the variable-task l' such that variable-task l' is executed on a processor at slot 3,
– X_2 is the set containing the variable-task l' such that the variable-task l' is executed at slot 4 or after,
– X_3 is the set containing the variable-task l' such that the variable-task $\bar{l}'$ is executed at slot 3,
– X_4 is the set containing the variable-task l' such that the variable-task$\bar{l}'$ is executed at slot 4 or after,
– X_5 is the set containing the variable-task l' such that the variable-task $\hat{l}'$ is executed at slot 2 or 3,
– X_6 is the set containing the variable-task l' such that the variable-task $\hat{l}'$ is executed at slot 4 or after.

Let be $x_i = |X_i|$ for $i \in \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$.
We can stem from the construction of an instance of the scheduling problem,
– $x \leq \frac{n}{3}$ (there are only $\frac{n}{3}$ clauses-tasks C_i),
– $x_2 + x_4 \leq n$ (there are only n processors which executed a clause-task D_5^i),
– $x_3 + x_5 \leq n$ ($\forall\ x'$, $\hat{x}'$ and $\bar{x}'$ can not be executed together at slot 3 in the same processor (they have a common predecessor)),
– $x_6 \leq \frac{2n}{3}$ (there are at most $\frac{n}{3}$ processors which executed a clause-task A^i and each processor have only the slot 4 and 5 to execute variables-tasks $\hat{x}'$).

So we can deduce $\sum_{i=1}^{6} x_i \leq 3n$. As $\sum_{i=1}^{6} x_i = 3n$ (all the variables-tasks must be executed), the inequalities are all equalities. And finaly we have $x_1 = x_4 = x_5 = \frac{n}{3}$ and $x_2 = x_3 = x_6 = \frac{2n}{3}$.
If the variable-task x' (resp. the variable-task $\bar{x}'$) is executed at slot 3, the variable-task $\bar{x}'$ (resp. x') must be executed at or after slot 4. Therefore, $X_1 \subseteq X_4$ and $X_3 \subseteq X_2$ and finaly as the cardinalities are equal $X_1 = X_4$ and $X_3 = X_2$.
So if we affect the value "true" to the variable l if and only if the variable-

task l' is executed at slot 3 is trivial to see that in the clause of length 3 we have one and only one literal equal to "true".
Let be $c = (x \vee \bar{y})$, a clause of length 2. As in the preuve of the theoreme 1 :
– If $x' \in X_1 \implies y' \in X_4 \implies y' \in X_1$. Only the literal x is "true" between x and $\bar{y}$,
– If $x' \in X_2 \implies y' \in X_3 \implies y' \in X_2$. Only the literal $\bar{y}$ is "true" between x and $\bar{y}$

- Conversely, we suppose that there is a truth assignment $I : \mathcal{V} \to \{0, 1\}$, such that each clause in $\mathcal{C}$ has exactly one true literal.
  Suppose that the true litteral in the clause $C_i = (y \vee z \vee t)$ is t. Therefore, the variable-task t' is processed at the slot 3 on the the same processor which execute the clause-task C^i and the task α'_t. The task $\bar{t}'$ (resp. y', z') are executed on the processor which executed the clause-task D_5^i corresponding to the clauses of length 2 where $\bar{t}$ appears (resp. y, z). The clause-task $D_j^i,\ j = 1, 2, 3, 4$, is executed at slot j on the same processor. The $\frac{2n}{3}$ other variables-tasks $\bar{y}'$ not yet schedule are executed at slot 3 in the processor which execute $\alpha_{y'}$ and $\beta_{y'}$. The variables-tasks $\hat{t}'$ (resp. $\hat{y}'$ and $\hat{z}'$) are executed at slot 2 (resp. 4 and 5) on the processor executing β'_t (resp. A^i). We can observe that this scheduling is valid.
  This concludes the preuve of the theoreme 5.4.2.

□

Corollaire 5.4.1 *There is no polynomial-time algorithm for the problem* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{max}$ *with performance bound smaller than* $1 + \frac{1}{c+4}$ *unless* $\mathcal{P} \neq \mathcal{NP}$.

Preuve

The preuve of Corollaire 5.4.1 is an immediate consequence of the Impossibility Theoreme, (see [8], [10]).

□

5.5 A polynomial time for $C_{max} = c + 2$ with $c \in \{2, 3\}$

Théorème 5.5.1 *The problem of deciding whether an instance of* $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max}$ *with* $c \in \{2, 3\}$ *has a schedule of length at most* $(c+2)$ *is solvable in polynomial time.*

Preuve

For $c = 2$. It easy to see that the source (resp. the sink) is executed at the slot 0 (resp. the slot $c + 2$). If a source i (resp. a sink i) is scheduled before or at the slot

2 (resp. before or at the slot $c+1$) then the task i admits only one successor (resp. only one predecessor). The others tasks must be executed as soon as possible.

□

Remark : We conjecture that the problem of deciding whether an instance of $\bar{P}|prec; c_{ij} = c; p_i = 1|C_{max}$ with $c \geq 2$ has a schedule of length at most $(c+3)$ is solvable in polynomial time.

5.5.1 Approximation

In this section, we developp a new method based of a notion of expansion of a schedule. Let us first give some notation before an explanation of the method.

Notation : We denote by σ^{∞}, the UET-UCT schedule, and by σ_c^{∞} the UET-LCT schedule. Moreover, we denote by t_i (resp. t_i^c) the starting time of the task i in the schedule σ^{∞} (resp. in the schedule σ_c^{∞}).

Principle :

An idea consists to keep an anssignment for the tasks given by a "good" feasible schedule on unbounded number of processors σ^{∞}, and proceed to an expansion of the makespan, in order to temporal distance between two tasks, i and j with $(i,j) \in E$, processing on two differents processors respect the communications delays *i.e.* $t_j^c \geq t_i^c + 1 + c$.

5.5.2 Description of the method

Let be a precedence graph $G = (V, E)$, we determinate a feasible schedule σ^{∞}, for the model UET-UCT, using an $(4/3)-$approximation algorithm proposed by Munier and König [24]. This algorithm gives a couple $\forall i \in V,\ (t_i, \pi)$ on the schedule σ^{∞} corresponding to :

– t_i the starting time of the task i for the schedule σ^{∞} and

– π the processor on which the task i is processed at t_i.

Now, we determinate a couple $\forall i \in V,\ (t_i^c, \pi')$ on the schedule σ_c^{∞} in the following ways :

– The starting time $t_i^c = d \times t - i = \frac{(c+1)}{2} t_i$ and,

– $\pi = \pi'$.

The justification of expansion coefficient is given below. An illustration of expansion is given by the Figure 5.3.

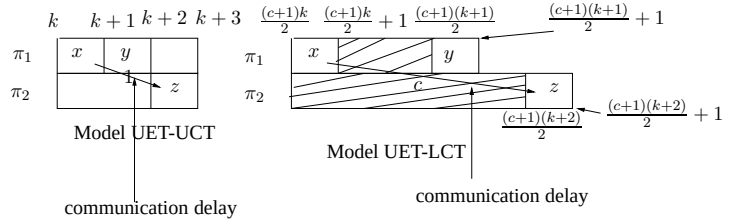


FIG. 5.3 – *Illustration of notion of an expansion*

5.5.3 Analysis of the method

Lemme 5.5.1 *The coefficient of an expansion is* $d = \frac{(c+1)}{2}$.

Preuve Let be two tasks i and j such that $(i,j) \in E$, which are processed on two differents processors in the feasible schedule σ^{∞}. We are interesting in coefficient d such that $t_i^c = d \times t_i$ and $t_j^c = d \times t_j$. After an expansion, in order to respect the precedence constraints and the communication delays we must have $t_j^c \geq t_i^c + 1 + c$, and so

$$\begin{aligned} d \times t_i - d \times t_j &\geq c+1 \\ d &\geq \frac{c+1}{t_i - t_j} \\ d &\geq \frac{c+1}{2} \end{aligned}$$

It sufficient to choose $d = \frac{(c+1)}{2}$.

□

Lemme 5.5.2 *An expansion algorithm gives a feasible schedule for the problem denoted by* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{\max}$.

Preuve It sufficient to check that the solution given by an expansion algorithm produces a feasible schedule for the model UET-LCT. Let be two tasks i and j such that $(i,j) \in E$. We denote by π_i (resp. π_j) the processor on which the task i (resp. the task j) is executed in the schedule σ^{∞}. Moreover, we denote by π_i' (resp. π_j') the processor on which the task i (resp. the task j) is executed in the schedule σ_c^{∞}. Thus

– If $\pi_i = \pi_j$ then $\pi'_i = \pi'_j$. Since the solution given by Munier and König gives a feasible schedule on the model UET-UCT, then we have

$$\begin{aligned} t_i + 1 &\leq t_j \\ \frac{2}{c+1}t_i^c + 1 &\leq \frac{2}{c+1}t_j^c \\ t_i^c + 1 \leq t_i^c + \frac{c+1}{2} &\leq t_j^c \end{aligned}$$

– If $\pi_i \neq \pi_j$ then $\pi'_i \neq \pi'_j$. Since the solution given by Munier and König gives a feasible schedule on the model UET-UCT, then we have

$$\begin{aligned} t_i + 1 + 1 &\leq t_j \\ \frac{2}{c+1}t_i^c + 2 &\leq \frac{2}{c+1}t_j^c \\ t_i^c + (c+1) &\leq t_j^c \end{aligned}$$

□

Théorème 5.5.2 *An expansion algorithm gives a* $\frac{2(c+1)}{3}$*−approximation algorithm for the problem* $\bar{P}|prec; c_{ij} = c \geq 2; p_i = 1|C_{\max}$.

Preuve We denote by C^h_{max} (resp. C^{opt}_{max}) the makespan of the schedule computed by the Munier and König (resp. the optimal value of a schedule σ^∞).

In the same way we denote by $C^{h^*}_{max}$ (resp. $C^{opt,c}_{max}$) the makespan of the schedule computed by our algorithm (resp. the optimal value of a schedule σ^∞_c).

We know that $C^h_{max} \leq \frac{4}{3}C^{opt}_{max}$. Thus, we obtain

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

□

Remark : The method of an expansion can be used for another problems.

5.6 Analysis of our results

With the result proposed in this article, we complete the table 5.1. We proved that in the first case, as we known, that the lower bound of approximation is the same for the problems with bounded and unbounbed number of processors in presence of large communications delays.

Type of machines	c_{ij}	C_{max}	Complexity	Lower bound	References
$\bar{P}$	$c = 1$	5	Polynomial		[31]
$\bar{P}$	$c = 1$	6	$\mathcal{NP}$-complete	$7/6 \leq \rho$[1]	[31]
P	$c = 1$	3	Polynomial		[26]
P	$c = 1$	4	$\mathcal{NP}$-complete	$5/4 \leq \rho$[2]	[31]
$\bar{P}$	$c \in \{2, 3\}$	$c + 2$	Polynomial		5.5.1
$\bar{P}$	$c \geq 2$	$c + 4$	$\mathcal{NP}$-complete	$1 + 1/(c + 4) \leq \rho$[3]	5.4.1
$\bar{P}$	$c \geq 2$	$c + 1$	Polynomial		[5]
$\bar{P}$	$c \geq 2$	$c + 3$	$\mathcal{NP}$-complete	$1 + 1/(c + 3) \leq \rho$[4]	[5]

TAB. 5.2 – Complexity results

Type of machines	c_{ij}	Approximation results	References
P	$c = 1$	$\rho \leq 4/3$	[21]
P	$c = 1$	$\rho \leq 7/3$	[20]
$\bar{P}$	$c \geq 2$	2 for tree	[19]
$\bar{P}$	$c \geq 2$	$\frac{2(c+1)}{3}$	5.5.2
P	$c \geq 2$	$O(c)$	[27]

TAB. 5.3 – New approximation results

The new complexity results are given by the Table 3.

[1] $3SAT$ see[10]
[2] $clique$ see[10]
[3] Π_1 see Therorem 5.4.1
[4] $BBCG$ see [10, 29]

CHAPITRE 6

Conclusion

Dan ce cours, nous avons quelques résultats en ordonnancement avec ou sans communication. Il existe d'autres modèles qui prennent en compte la vitesse de traitement des processeurs, des processeurs sont regroupés sous forme de modules de processeurs avec plusieurs niveaux de communications, d'autres critères à minimiser ou à maximiser ....

La théorie de l'ordonnancement est un domaine vaste et la communauté reste très active.

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] 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.

[3] A. Gérasoulis ans T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transaction on Parallel distributed systems*, 4 :686–701, 1993.

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

[5] 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.

[6] 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.

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

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

[9] 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.

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

[11] 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.

[12] 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. *Ann. Discrete Math.*, 5 :287–326, 1979.

[13] 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.

[14] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *O. R. Lett.*, 16(3) :129–137, 1994.

[15] 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.

[16] J.K. Lenstra, M. Veldhorst, and B. Veltman. The complexity of scheduling trees with communication delays. *Journal of Algorithms*, 20 :157–173, 1996.

[17] 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.

[18] 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.

[19] A. Munier. Approximation algorithms for scheduling trees with general communication delays. *Parallel Computing*, 25(1) :41–48, January 1999.

[20] 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.

[21] A. Munier and C. Hanen. An approximation algorithm for scheduling unitary tasks on m processors with communication delays. Private communication, 1996.

[22] A. Munier and C. Hanen. Using duplication for scheduling unitary tasks on m processors with communication delays. *Theoretical Computer Science*, 178 :119–127, 1997.

[23] A. Munier and C. Hanen. Performance of Coffman-Graham schedules in the presence of unit communcation delays. *Discrete Applied Mathematics*, 81 :93–108, 1998.

[24] A. Munier and J.C. König. A heuristic for a scheduling problem with communication delays. *Operations Research*, 45(1) :145–148, 1997.

[25] C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comp.*, 19(2) :322–328, April 1990.

[26] C. Picouleau. New complexity results on scheduling with small communication delays. *Discrete Applied Mathematics*, 60 :331–342, 1995.

[27] 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.

[28] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discr. App. Math.*, 18 :55–71, 1987.

[29] R. Saad. Scheduling with communication delays. *JCMCC*, 18 :214–224, 1995.

[30] J.D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10 :384–393, 1975.

[31] B. Veltman. *Multiprocessor scheduling with communications delays*. PhD thesis, CWI-Amsterdam, Holland, 1993.