Historique des travaux autour du problème P||Cmax

# Introduction

# Présentation du problème.

## Parallélisme.

Le parallélisme est un type d'architecture informatique dans lequel plusieurs processeurs exécutent ou traitent une application ou un calcul simultanément. IL aide à effectuer de grands calculs en divisant la charge de travail entre plusieurs processeurs, qui fonctionnent tous en même temps.

Il existe quatre types de parallélismes, définis par la taxonomie de Flynn(1). Cette classification est basée sur deux notions : le flot d’instructions (simple ou multiple), et le flot de données (simple ou multiples) ; un algorithme est un flot d’instructions à exécuter sur un flot de données.

|  |  |  |
| --- | --- | --- |
| Instructions / Données | Simple | Multiple |
| Simple | SISD  premiers PC  machine de Von Neumann  Obsolète, car tous les PC sont désormais multi-cœur. | SIMD  Machines synchrones  Pipeline  Exécution d’une instruction unique sur des données différentes. |
| Multiple | MISD  Machines vectoriels  Tableau de processeurs  Exécute plusieurs instructions sur une même donnée. | MIMD  Multi processeurs à mémoire distribuée.  Multi processeurs à mémoire partagée (multi-cœur).  Multi Ordinateur. |

Taxonomie de Flynn

Les premières machines parallèles étaient des réseaux d’ordinateurs, et des machines vectorielles (faiblement parallèles, très coûteuses), telles que l’IBM 360, les Cray1. La plupart des machines parallèles contemporaines sont désormais MIMD.

On peut définir une machine parallèle comme un ensemble de processeurs qui coopèrent et communiquent.



IBM 360-91 (le plus rapide et le plus puissant en service en 1968) NASA. Centre de vols de Greenbelt (Md)

## Ordonnancement

Sur une machine non parallèle, les tâches sont exécutées séquentiellement, les unes après les autres. Certaines tâches, ou jobs peuvent demander plus de temps que d’autres pour être entièrement traitées.

Lorsque plusieurs ressources (processeurs, machines, cœurs) sont disponibles, ou que des jobs a exécuter ne sont pas indépendants (même traités sur un seul processeur), se pose alors, un problème d’ordonnancement.

Celui-ci consiste à organiser, dans le temps, les jobs à exécuter, en les affectant à une ressource donnée, de manière à satisfaire un certain nombre de contraintes, tout en optimisant un ou des objectifs.

L’ordonnancement, fait partie de la catégorie des problèmes d’optimisation combinatoire.

Les problèmes qui s’y rattachent sont très variés.

Premièrement, la nature des machines parallèles doit être considérée. Celles-ci peuvent être identiques (Le même temps de traitement sera nécessaire, d’une machine à l’autre) ; uniformes (un quotient de vitesse qi propre à une machine est à appliquer pour chaque tâche affectée à cette machine pour déterminer le temps de traitement nécessaire) ; ou indépendantes (les temps de traitements des tâches sont ni uniformes ni proportionnels d’une machine à l’autre).

Ensuite, des contraintes peuvent affecter les jobs eux-mêmes. Dans le cas d’un problème préemptif, les taches peuvent être interrompues, et reprises ultérieurement. Il est possible que les jobs soient indépendants, ou au contraire, être liées par des relations de précédence. Ces jobs ne sont disponibles qu’à partir d’une certaine date. Ou encore, être de durée égale, ou tous de durée différente.

Pour finir, l’objectif de l’ordonnancement est d’optimiser un critère. Par exemple, minimiser la somme des dates de fin, la somme des retards, le nombre de tâches en retard, ou simplement, le retard total. Mais le plus habituel, est de chercher à minimiser le temps total de traitement de tous les jobs, i.e minimiser le makespan.

## Enoncé du P||Cmax

Ces diverses possibilités définissent divers problèmes d’ordonnancements différents, recensés et classifiés par Graham et al. [1], qui introduit la notation trois-champs α|β|γ .

Le problème Pm||Cmax se définit alors ainsi :

* α = α1 α2, détermine l’environnement machines.

α = P : Les machines sont parallèles et identiques : Un job, une tâche prendra le même temps de traitement qu’il soit exécuté sur une machine ou une autre. Le nombre de machine (m) est variable.

* β c { β1, β2, β3, β4, β5, β6}, détermine les caractéristiques des jobs, ou des tâches.

β est vide. Ce qui signifie que la préemption n’est pas autorisée (les jobs doivent être exécutés d’une traite, sans interruption ni coupure) et qu’il n’y a pas de relation entre les jobs (ils sont indépendants).

* γ détermine le critère à optimiser.

γ = Cmax : on cherche à optimiser le makespan, i.e le temps de traitement total.

Pm||Cmax consiste à planifier un ensemble J = {1,2,…,n) de n jobs simultanés, pour être traités par m machines identiques et parallèles. Chaque job, qui requière une opération, peut être traité par une des m machines. Le temps de traitement de chaque job (Pi avec i ∈ N) est connu à l’avance. Un job commencé, et complété sans interruption. Les jobs, indépendants, sont exécutés par une seule machine, et une machine ne peut traiter qu’un seul job à la fois.

## Problématique

Comme l’ont démontré Garey et Johnson, P2||Cmax est un problème NP-Difficile [4], et P||Cmax est un problème NP-Difficile au sens fort [3]. Cependant, Pm||Cmax devient un problème NP-Difficile, du moment que le nombre de machines est fixé [2], comme l’a montré Rothkopf [5], qui a présenté un algorithme de programmation dynamique.

Donner la solution optimale à un problème d’ordonnancement (dans notre cas Pm||Cmax n’est pas réaliste. Même pour un problème de taille modeste, la résolution de celui-ci demanderait un temps excessif et donc rédhibitoire.

La résolution du problème d’ordonnancement va reposer sur des méthodes d’approche, qui consistent à calculer en temps polynomial, une solution « assez » proche de la valeur optimale.

Dans la littérature, l’étude d’ordonnancement est très riche et abondante. Le but étant d’améliorer le temps de calcul, et d’approcher le résultat optimal.

# Résoudre le problème

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Phrase choc d’introduction

Comme évoqué précédemment, l’existence d’une solution qui résout le problème n’est // // pas pensable (à moins que P=NP …). De fait, les solutions proposées en temps polynomial // sont approchées. Il est impossible de présenter une liste exhaustive des solutions proposées tant la recherche dans ce domaine est soutenue. .

Il existe plein de solutions blabla LS, … Reseaux de neurones, genetique…

Petit plan

Ici on presente les methodes les plus utilisées, et quelques algo par methode.

Après un rappel des notations utilisées, methode basée sur LP, des heuristiques basées sur LS, Bin pacqking, pour finir avec un PTAS, et un partitionnement.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Notations utilisées

Chaque document utilise sa propre notation, mais les notions sont les mêmes.

Soient les données du problème Pm||Cmax,

Un ensemble de n jobs J = {1,2,…,n)

Chaque job j a un temps de traitement pj P= {p1, p2, …, pn}

m machines parallèles identique Mi avec (i=1,2,…,m)

(J) : le résultat de l’ordonnancement d’un ensemble J de jobs, sur m machines parallèles, identiques, obtenu par un algorithme A.

(J) : le makespan optimal.

r(A) = (J) / le ratio d’approximation atteint par l’algorithme A au pire cas.

## Heuristiques

Les heuristiques présentent plusieurs avantages. Leur complexité est réduite, et obtiennent de bonnes performances. Elles représentent la plus grande partie des recherches concernant le problème d’ordonnancement, même si leurs performances, au pire cas, n’est pas garantie.

Sont abordées ici les heuristiques les plus présentes dans la littérature.

### Basé LS (List scheduling)

L’idée d’une LS est de stocker l’ensemble des jobs dans celle-ci, les trier dans un ordre particulier, avant de les affecter à une machine selon des règles définies.

#### Algorithme LPT rule graham 1969

Graham propose [8] Longest Processing Time (LPT) rule.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Algorithme LPT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input : instance de Pm||Cmax, avec m machines, n jobs et leur temps d’exécution

1 : trie les jobs de l’ensemble J dans l’ordre décroissant de leur temps d’exécution et réindexe l’ensemble, de telle manière à obtenir :

P1 >= P2 >= … >= Pn

2 : Parcours la liste, et affecte chaque job à la machine la moins chargées, à ce moment-là.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exemple.

Soit P = (13,10,7,6,6,5,3,2) l’ensemble des Pj déjà triés dans l’ordre décroissant

à appliquer sur 4 machines parallèles identiques.



Nous obtenons (J) = 14

Le tri puis l’affectation s’effectuent en **O(n log n + n log m)**

Le ratio d’approximation **r(LPT) <= -**

#### Algorithme LPT-REV (Croce et Scatamacchia, 2018)

Ce ratio d’approximation obtenu par LPT (r(LPT) = - ) est une borne supérieure que LPT peut atteindre, mais qu’il ne dépassera jamais. Chaque utilisation de LPT donnera un résultat qui oscillera entre 1 et = - .

Exemple de pire cas.

Soit P = (7,7,6,6,5,5,4,4,4) l’ensemble des Pj déjà triés dans l’ordre décroissant

à appliquer sur 4 machines parallèles identiques.

* LPT donne le résultat suivant :



(J) = 15

* Un bon ordonnancement aurait donné :



(J) = 12

* Soit une marge d’erreur de 15/12

Pour m = 4

- = - =

Ce cas représente un pire cas pour LPT

Croce et Scatamacchia [9] en examinant le comportement de « LPT rule », notamment au niveau du ratio d’approximation, constatent que celui-ci peut être réduit selon certaines configurations, ou instances du problème, et rédigent le théorème suivant :

LPT a un rapport d'approximation non supérieur à - pour m ≥3 et n <> 2m + 1.

LPT atteint la limite de Graham de - pour m ≥2 et uniquement dans les cas où

n = 2m + 1, et la machine critique traite 3 jobs, tandis que les autres en traitent 2.

La machine critique est la machine qui exécute le job critique

Le job critique (noté J’) est le job qui détermine le makespan.

Le rapport d’approximation augmente moins vite (en fonction du nombre de machines) que le précédent à condition d’éviter certaines instances.

NB

L’exemple précédant (pire cas) a les caractéristiques suivantes :

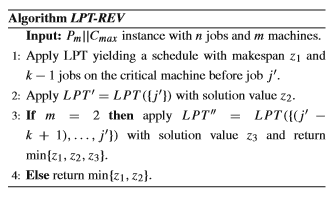
Nombre de jobs n = 2m +1

La machine critique exécute 3 jobs

Les autres exécutent 2 jobs

Un rapport d’approximation de -

Une modification à l’algorithme LPT est apportée pour placer le problème Pm||Cmax toujours dans une instance où le ratio d’approximation est - . Cette modification consiste à planifier en premier, le job critique sur une machine M1.



Le ratio d’approximation **r(LPT-REV) <= -**

### Basé Bin-Packing

Le problème Bin-packing, est semblable au problème Pm||Cmax. Il consiste à ranger des objets dans des bacs de taille similaires, tout en minimisant le nombre de boites.

L’ensemble des n jobs J = {1,2,…,n) et de leur temps de traitement pj P= {p1, p2, …, pn} peuvent être vus respectivement, comme :

un ensemble d’objets **T = {T1,T2, … ,Tn}**,

et leur taille **L(Ti)**

Une taille maximale **C** des bacs (ou boîtes) est donnée.

Un packing, est une partition P<P1, P2,…,Pm> de T telle que L(Pj) <= C (avec 1<=j<=m)

Le but est de placer les Ti dans des bacs Pj de taille C, de manière à minimiser le nombre de bacs m.

L’idée est d’utiliser le problème à l’envers, pour approcher une solution au problème d’ordonnancement.

#### Algorithme MULTIFIT

Coffman, Garey, et Johnson [10] se sont intéressés à l’algorithme FFD (First Fit Decreasing), un outil de résolution du problème « bin-packing », pour l’adapter au problème d’ordonnancement.

FFD(T,C) renvoie le nombre de bacs de taille C non vides nécessaires, et l’arrangement correspondant de l’ensemble T d’objets.

Soit = min{C :FFD(T,C) <= m} la plus petite valeur de C (taille des bacs) qui permet à T d’être pacqué dans m (ou moins) bacs.

Le but de Multifit, est donc, de réduire la valeur de C, faire tourner FFD(T,C), jusqu’à ce que le nombre m de bacs alors, devenu insuffisant, augmente à m+1.

Cette valeur charnière de C est , qui correspond au makespan minimum recherché, de l’ordonnancement de l’ensemble T de jobs sur m machines parallèles identiques.



Fonctionnement de FFD et principe de MULTIFIT

La recherche de s’effectue par dichotomie.

La borne supérieure Cu[T,m] = max{(2/m)\*L(T), maxi{L(Ti)}}

La borne inférieure Cl[T,m] = max{(1/m)\* L(T), maxi{L(Ti)}}

Multifit reçoit les paramètres suivants

T, un ensemble de jobs

m, un nombre de processeurs

k, un nombre d’itérations maximal (pour la recherche dichotomique)

Après k itérations, Multifit renvoie Cu(k) qui correspond à la plus petite valeur C pour laquelle FFD[T,C] <= m

Le tri puis k FFD s’effectuent en **O(n log n + kn log m)**

Ratio[11] **r(MF) <= 1.220 + 2-k**

Généralement, Multifit donne un résultat très satisfaisant avec k=7.

#### Algorithme COMBINE

Lee et Massey [11] ont l’idée d’utiliser LPT pour réduire les bornes de départ de Multifit dans un algorithme nommé COMBINE.

Soient la moyenne des poids des jobs par processeur A =

M = (J)

Et M\* = (J)

Si M >= 1.5 A alors M\* = M

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Algorithme COMBINE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input : instance de Pm||Cmax, avec m machines, n jobs et un coefficient d’arrêt α (0.005)

1 : A =

M 🡨 (J)

Si M >= 1.5 A alors M\* = M

Sinon aller en 2 :

2 : Appliquer Multifit avec

Cu = M

Cl = max{ ( M / ( - ) ), P1, A} (P1 : job le plus long)

Arrêter lorsque Cu – Cl <= αA

Complexité **O(nlogn + knLogm)**

Ratio[12] **r(CB) <= 13/11 + 2-k**

Avec k le nombre d’itérations de recherches dichotomiques.

Concernant la complexité

Généralement les itérations Combine k = 6 (lorsque Cu – Cl <= αA), mais il a déjà exécuté une fois LPT, donc, généralement, k = 7.

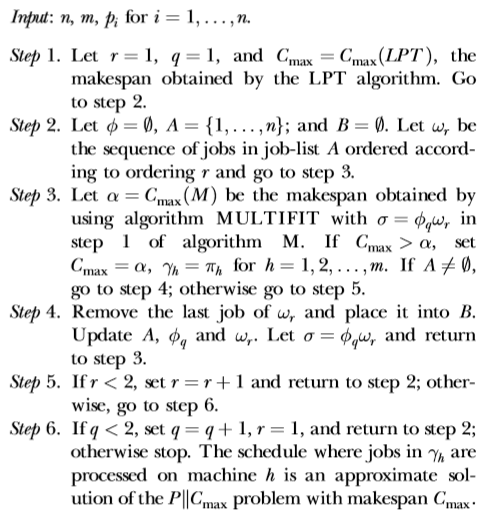
#### Algorithme LISTFIT

Gupta et Ruiz-Torres [12], ont aussi l’idée d’utiliser Multifit, afin de réaliser l’algorithme Listfit.

Celui-ci sépare la liste des travaux en 2 sous-listes, traitées soit dans un ordre LPT (longest Time Processing), soit dans un ordre SPT (Shortest Time Processing). Puis, Listfit combine ces deux sous-listes en appliquant MultiFit à chaque itération.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Algorithme Listfit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Complexité **O(n2 log n + kn2blog m)**

Ratio **r(MF) <= 13/11 + 2-k**

Où k est le nombre d’itérations pour Multifit.

### Aproche gloutonne

#### Algorithme SLACK (Croce et Scatamacchia, 2018)

Croce et Scatamacchia [9], en effectuant la preuve d’une borne d’approximation pour le développement de LPT-Rev, ont mis en évidence l’importance des différences de temps entre les jobs, ainsi que le regroupement de ceux-ci en sous-ensembles.

Notamment, pour l’instance suivante

Nombre de jobs n = 2m+1

Avec P2m+1 >= P1 - Pm.

Où ils ont planifié le job 2m+1, puis le sous ensemble trié {1 … m}, puis le sous ensemble trié {m+1 … 2m}.

En résulte la stratégie gloutonne suivante

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Algorithme SLACK

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 : Trier la liste des jobs dans l’ordre décroissant des temps nécessaires de traitement

2 : Réindexer les jobs, de manière à obtenir P1 >= P2 >= … >= Pn

3 : Découper l’ensemble en n/m tuples de m Jobs (ajout de jobs « dummy » de taille nulle pour le dernier tuple, si n n’est pas un multiple de m).

4 : Considérer chaque tuple avec la différence de temps entre le premier job du tuple, et le dernier, appelée Slack.

{ { 1, … , m} {m+1 , … , 2m} … }

P1 – Pm Pm+1 – P2m …

5 : Trier les tuples par ordre décroissant de Slack, et ainsi former un nouvel ensemble

{{m+1 , … , 2m} { 1, … , m}} (si Pm+1 – P2m > P1 – Pm)

6 : Appliquer L’ordonnancement (Affectation à la machine la moins chargée à ce moment-là) à l’ensemble ainsi obtenu

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Programmation Linéaire

L’ordonnancement, et plus particulièrement Pm||Cmax s’inscrit parfaitement dans l’énoncé d’un problème de programmation linéaire. En effet la fonction objectif, minimiser le makespan, ainsi que les contraintes sont des fonctions linéaires.

Toutefois, les variables, et le résultat attendus sont discrets, ce qui rend la résolution du problème nettement plus difficile comparé à une programmation linéaire à variables continues.

Ces algorithmes donnent une solution faisable, exacte.

#### Algorithme « PA » de Mokotoff.

Makatoff [6] présente un algorithme basé sur la formulation de la programmation linéaire, en utilisant des variables booléennes d’affectation des jobs à une machine.

La minimisation du makespan peut être posée ainsi :

Min y tel que

= 1 pour 1<= i <= n

Sur toutes les machines, au moins un et un seul xi est égal à 1.

Un job est affecté à une et une seule machine.

Y - pour 1<= j <= m

Pour une machine donnée, la somme des temps <= y

Où la valeur optimale de y est le Cmax

et Xij = 1 si le job i est affecté à la machine j

= 0 si le job i n’est pas affecté à la machine j

Le programme linéaire est donc composé de

n\*m + 1 variables (les variables xij et la variable y)

n+m contraintes

La zone F peut être définie ainsi :

F = { (x,y) : x Bn\*m, y +: = 1 ; Y - , }

avec B =

Le polytope P, relatif à F est défini ainsi

P = { (x,y) : x R+n\*m, y + : = 1 ; Y - , }

Il est possible de construire un ensemble fini d’**inégalités**

Ax + Dy <=  telles que min{y : (x,y) F} = min{y : x R+n\*m , y R+, Ax+Dy <= }

NB : Une solution (x°,y°) P doit être exclue (n’est pas un vecteur entier) si (x°,y°) ∉ F

Des **Inégalités transitoires** peuvent être générées ( nombre maxi de job par machine)

<= Lj (Lj = h-1 ⬄ Sjh > Lb et Sj(h-1) <= Lb)

Lb : Borne inférieure

Pour un problème Pm||Cmax, même de taille modeste, le nombre de variables, contraintes, et très important, dont certaines sont inutiles. L’algorithme va donc utiliser la méthode des plans sécants (Cutting Planes Technique). A chaque itération, des inégalités valides sont générées, puis une relaxation est exécutée. Jusqu’à l’obtention d’une solution faisable.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Algorithme PA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Détermination de la borne inférieure (Lb) suivant l’algorithme de McNaughton [7].

Détermination de la borne supérieure Ub (juste pour la nommer) suivant l’heuristique LPT.

Si Lb coïncide avec Ub, alors la solution optimale est trouvée

Sinon, le processus itératif démarre :

Dans chaque itération, un programme de relaxation linéaire est résolu dans lequel Cmax doit être égal à la borne inférieure actuelle Lb.

Si la solution obtenue est entière, donc faisable, l'algorithme s'arrête et la solution actuelle est optimale.

Sinon, des nouvelles inégalités (inégalités et/ou inégalités transitoires), sont ajoutées à la relaxation linéaire. Le nouveau programme linéaire est résolu et l'algorithme s'arrête si la solution est entière.

Si la relaxation n'est pas possible, la limite inférieure Lb est augmentée d’une unité et le processus, redémarre.

Par contre, Si les inégalités ne peuvent pas être générées, un algorithme Branch&Bound prend le relais pour résoudre le problème.

## Approximation

Une catégorie d’algorithmes fournit une garantie d’approche. C’est le cas, notamment, des PTAS (Schéma d’Approximation en Temps Polynomial).

Principe PTAS (programmation dynamique, approximation ε …

* Algorithme « Using dual approximation Algorithm for Scheduling problems : Theorical and Practical Results » (Hochbaum et Shmoys 1987)

## Autres approches

#### LDM

Transition

LPT fait toujours référence pour comparer chaque algorithme développé…

..

Les heuristiques, font plus l’objet de recherches ….

# Synthèse

Récapitulatif complexité / Borne d’approximation… théoriques

Quelques résultats comparés expérimentaux

Avantages inconvénients

# Conclusion

* 1. Utilisation des algorithmes

Slack est utilisé dans …

* 1. Point de vue personnel

1. Recherche documentaire

Document LDM n’explique pas comment les partitions sont construites (semble utiliser LPT)

Difficulté de trouver des applications aux algorithmes

Beaucoup de documents indisponibles, payant…

Acteurs prolifiques Graham, Mokotoff, …

* 1. Futur ? …

Remarques

(1). A été trouvé aussi sous le nom de « taxonomie de Tanenbaum »

Un site effectue régulièrement, un top 500 des machines les plus puissantes :

<https://www.top500.org/>

Des sites internet rassemblent des documents concernant les problèmes d’ordonnancement :

<http://www.mathematik.uni-osnabrueck.de/research/OR/class/>

<http://schedulingzoo.lip6.fr/>

# Références

[1]

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979).Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, & B. H. Korte (Eds.), Discrete optimization II, annals of discrete mathematics (Vol. 5, pp. 287–326).

[2]

Chen B.,Potts C.N.,and Woeginger G.J.(1999).Areviewofmachine scheduling: Complexity, algorithms and approximability. In D. Z. Du&P.M.Pardalos(Eds.),Handbookofcombinatorialoptimization: Volume 1–3. New York: Springer.

[3]\*

M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the TheoryofNP-Completeness,Freeman,SanFrancisco,1979.

[4]\*

M.R. Garey and D.S. Johnson, Strong NP-completeness results: motivation, examples and implications, Journal of the Association for Computing Machinery 25 (1978), 499-508.

[5]\*

M.H. Rothkopf, Scheduling independent tasks on parallel processors, Management Science 12 (1966), 437-447.

[6]

E. Mokotoﬀ, Scheduling to minimize the makespan on identical parallel machines: An LP-based algorithm, Investigacioon Operativa 8(1999)97–108.

[7]

McNaughton, R., "Scheduling with deadlines and loss function", Management Science 6, 1959, 1-12.

[8]

R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System TechnicalJournal 45 (1966), 1563-1581.

[9]

F. Della Croce and R. Scatamacchia, “The Longest Processing Time rule for identical parallel machines revisited,” Journal of Scheduling, 2018.

[10]

Coffman E.G Jr., Garey M. R., & Johnson, D. S. (1978). An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing, 7, 1–17.

[11]

Lee, C. Y., & Massey, J. D. (1988). Multiprocessor scheduling: Combining LPT and MULTIFIT. Discrete Applied Mathematics, 20(3), 233–242.

[12]

Gupta J. N. D., & Ruiz-Torres, A. J. (2001). A listﬁt heuristic for minimizing makespan on identical parallel machines. Production Planning & Control, 12(1), 28–36.