Historique des travaux autour du problème P||Cmax

florian colas

14 juin 2020

Sommaire

1 Introduction

texte

2 Présentation du problème.

texte

2.1 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.

Données / Instructions	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)

2.2 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, coeurs) 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) ;

— 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.

2.3 Enoncé du P||C $_{max}$

Ces diverses possibilités définissent divers problèmes d'ordonnancements différents, recensés et classifiés par Graham et al. [1], qui introduit la notation trois-champs $\alpha|\beta|\gamma$.

Le problème $P_m||C_{max}$ se définit alors ainsi :

— $\alpha = \alpha 1\ \alpha 2$, détermine l'environnement machines. $\alpha = P$: 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 machines (m) est variable.

— $\beta \subset \{\ \beta 1, \beta 2, \beta 3, \beta 4, \beta 5, \beta 6\}$, 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. $\gamma = C_{max}$: on cherche à optimiser le makespan, i.e le temps de traitement total.

Definition 1. $P_m||C_{max}$

$P_m||C_{max}$ 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 (P_i avec $i \in N$) 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.

2.4 Problématique

Comme l'ont démontré Garey et Johnson, $P_2||C_{max}$ est un problème NP-Difficile [4], et $P||C_{max}$ est un problème NP-Difficile au sens fort [5]. Cependant, $P\ _m||C_{max}$ devient un problème NP-Difficile, du moment que le nombre de machines est fixé [1], comme l'a montré Rothkopf [12], qui a présenté un algorithme de programmation dynamique.

Donner la solution optimale à un problème d'ordonnancement (dans notre cas $P\ _m||C_{max}$) 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.

3 Résoudre le problème

Comme évoqué précédemment, l'existence d'une solution qui résout le problème n'est pas pensable, à moins que P = NP.

3.1 Notations utilisées

Chaque document utilise sa propre notation, mais les notions sont les mêmes. Soient les données du problème

— un ensemble de n jobs (ou tâches) $J = \{1, 2, ..., n\}$ Chaque job j a un temps de traitement connu $p_{\mathrm{j}} P = \{p_1, p_2, ..., p_{\mathrm{n}}\}$

— m machines parallèles identiques M_{i} *avec* $(i = 1, 2, ..., m)$

— $C_j^A(J)$ Le résultat de l'ordonnancement d'un ensemble J de jobs, sur m machines parallèles, identiques, obtenu par l'algorithme A.

— $C_j^\star(J)$ Le makespan optimal, idéal.

— $\Gamma(A) = \dfrac{C_j^A(J)}{C_j^\star(J)}$ Le ratio d'approximation atteint par l'algorithme A au pire cas.

3.2 Heuristiques

Les heuristiques présentent plusieurs avantages. Leur complexité est réduite, et obtiennent de bonne performances. Elles représentent la plus grande partie des recherches concernant le problème 'ordonnancement, même si leurs performances, au pire cas, ne sont pas garanties. Sont abordées ici les heuristiques les plus présentes dans la littérature.

3.2.1 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.

LPT rule (Graham *et al.*, 1969)

Graham propose [6] ***Longest Processing Time (LPT) rule***.

Algorithm 1: LPT Rule

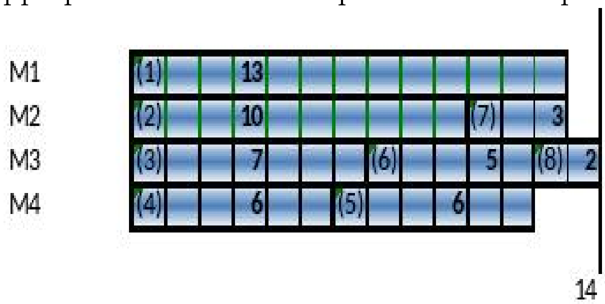
Data: instance de $\mathrm{P_m}||\mathrm{C_{max}}$, avec m machines, n jobs et leur temps d'exécution

Trie les jobs de l'ensemble J dans l'ordre décroissant de leur temps d'exécution et ré-indexe l'ensemble de telle manière à obtenir : $p_1 \geq p_2 \geq ... \geq p_n$

Parcours la liste, et affecte chaque job à la machine la moins chargée, à ce moment là.

Exemple

Soit $P = \{13, 10, 7, 6, 6, 5, 3, 2\}$, l'ensemble des p_j déjà triés dans l'ordre décroissant à appliquer sur 4 machines parallèles identiques :



Nous obtenons $C_4^{lpt}(J) = 14$

Le tri puis l'affectation s'effectuent en	$O(nlogn + nlogm)$
Le ratio d'approximation	$\Gamma(LPT) \leq \frac{4}{3} - \frac{1}{3m}$

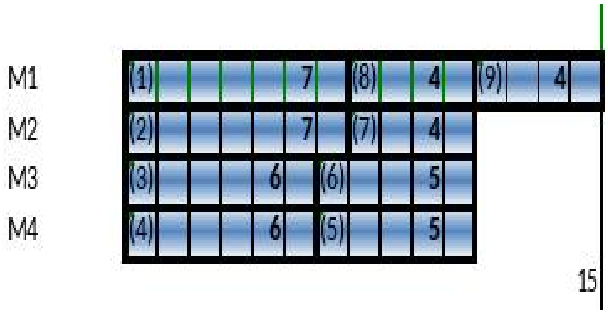
LPT-REV (Croce *et al.*, 2018)

Le ratio d'approximation obtenu par LPT (1) ($\Gamma(LPT) \leq \frac{4}{3} - \frac{1}{3m}$) est une borne supérieure que cet algorithme peut atteindre, mais qu'il ne dépassera jamais. Chaque utilisation de LPT produira un résultat dont le ratio Γ oscillera entre 1 et $\frac{4}{3} - \frac{1}{3m}$).

Exemple de pire cas

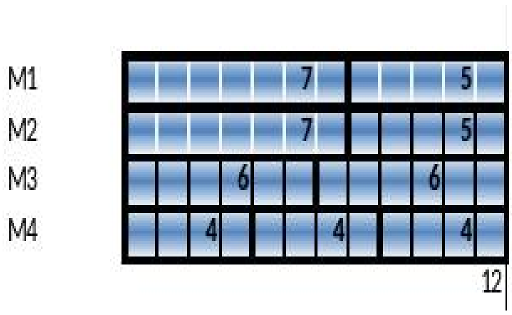
Soit $P = \{7, 7, 6, 6, 5, 5, 4, 4, 4\}$, l'ensemble des p_j déjà triés dans l'ordre décroissant à appliquer sur 4 machines parallèles identiques :

LPT donne le résultat suivant



$C_4^{lpt}(J) = 15$

Un ordonnancement optimal aurait été :



$C_4^{\star}(J) = 12$

Soit une marge d' de $\frac{15}{12}$

Le ratio d'approximation prévu pour $m = 4$ $\frac{4}{3} - \frac{1}{3m} = \frac{16}{12} - \frac{1}{12} = \frac{15}{12}$

Ce cas, représente donc un pire cas pour LPT.

Croce *et al.* [3], en examinant le comportement de LPT rule, notamment au niveau du ratio d'approximation, constatent qu'icelui peut être réduit selon certaines configurations, ou instances du problème, et rédigent le théorème suivant :

Théorème 1. *LPT a un rapport d'approximation non supérieur à* $\frac{4}{3} - \frac{1}{3(m-1)}$ *pour* $m \geq 3$ *et* $n \neq 2m + 1$.

LPT atteint la limite de Graham $\frac{4}{3} - \frac{1}{3m}$ pour $m \geq 2$ et uniquement dans le 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' $\frac{4}{3} - \frac{1}{3(m-1)}$ est inférieur au ration $\frac{4}{3} - \frac{1}{3m}$ (quel que soit le nombre de machines)

NB

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

- nombre de job $n = 2m + 1$.
- la machine critique exécute 3 jobs.
- les autres exécutent 2 jobs.
- un rapport d'approximation de $\frac{4}{3} - \frac{1}{3m}$

Une modification à l'algorithme LPT rule est apportée afin de placer le problème $\mathrm{P_m}||\mathrm{C_{max}}$ toujours dans une instance où le ratio d'approximation est $\leq \frac{4}{3} - \frac{1}{3(m-1)}$. Cette modification consiste à planifier en premier, le job critique sur une machine M1.

Algorithm 2: LPT-Rev

Data: instance de $\mathrm{P_m}||\mathrm{C_{max}}$, avec m machines, n jobs

1. Apply LPT yielding a schedule with makespan z_1 and $k-1$ jobs on the critical macine before job J'
2. Apply $LPT' = LPT(J')$ with solution value z_2
3. **If** $m = 2$ **then** apply $LPT'' = LPT([(J' - k + 1), ..., J'])$ with solution value z_3 and **return** $min[z_1, z_2, z_3]$
4. **Else return** $min(z_1, z_2)$

Le ratio d'approximation $\Gamma(LPT{-}REV) \leq \frac{4}{3} - \frac{1}{3(m-1)}$

3.2.2 Basé Bin-Packing

Le problème Bin-packing, est semblable au problème $P_m||C_{max}$. Il consiste à ranger des objets de taille différentes, dans des bacs identiques, tout en minimisant leur nombre.

L'ensemble des n jobs $J = \{1, 2, ..., n\}$, et de leurs temps de traitement p_j $P = \{p_1, p_2, ..., p_n\}$, peuvent être vus respectivement comme :

— un ensemble d'objets $T = \{T_1, T_2, ..., T_n\}$

— leur taille $L(T_i)$

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

Definition 2. Packing

Un packing, est une partition $P < P_1, P_2, ..., P_m >$ tel que $L(P_j) \leq C$ avec $1 \leq j \leq m$. Le but est de placer les objets T_i dans des bacs P_j de taille C, de manière à minimiser le nombre de bacs m.

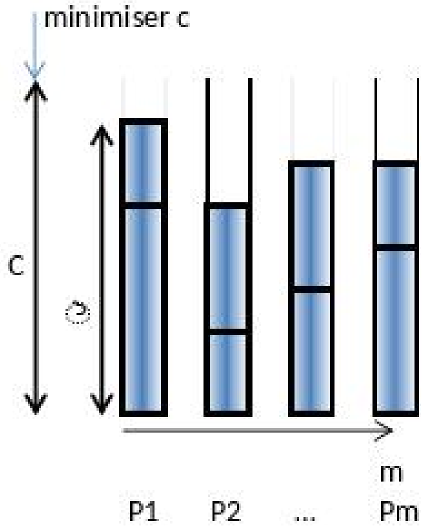
L'idée est d'utiliser le problème Bin-Packing à l'envers, pour approcher une solution au problème d'ordonnancement.

MULTIFIT

Coffman *et al.*, [2] se sont intéressés à l'algorithme FFD (First Fit Decreasing), un outil de résolution du problème de Bin-Packing, pour l'adapter au problème $P||C_{max}$. $FFD(T, C)$ renvoie le nombre de bacs de taille C non vides nécessaires, et l'arrangement correspondant de l'ensemble T d'objets.

Soit $T_m^\star = min\{C : FFD(T, C) \leq m\}$ 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 $T_m^\star$, qui correspond au makespan minimum recherché, de l'ordonnancement de l'ensemble T de jobs sur m machine identiques parallèles.



Fonctionnement de FFD et principe de MULTIFIT

Algorithm 3: MULTIFIT

Data: T un ensemble de jobs
m, un nombre de processeurs
borne supérieure : $Cu[T,m] = max\{\frac{2}{m} * L(T), max_i\{L(T_i)\}\}$
borne inférieure : $Cl[T,m] = max\{\frac{1}{m} * L(T), max_i\{L(T_i)\}\}$
k un nombre d'itérations

1 La recherche de $T_m^\star$ s'effectue par dichotomie sur k itérations

2 Après les k itérations, MULTIFIT **renvoie** $Cu(k)$ qui correspond à la plus petite valeur de C pour laquelle $FFD[T,C] \leq m$

Tri puis k FFD s'effectuent en	$O(nlogn + knlogm)$
Ratio [9]	$\Gamma(MULTIFIT) \leq 1,220 + 2^{-k}$

Généralement, MULTIFIT donne des résultats très satisfaisant avec $k = 7$

COMBINE

Lee *et al.*, [9] ont l'idée d'utiliser LPT (1) pour réduire les bornes de départ de MULTIFIT (3) dans un algorithme nommé COMBINE.

soient

la moyenne des poids des jobs par processeur $$A = \sum_{i=1}^{n} (\frac{P_i}{m})$$

et $$M = C_m^l pt(J)$$

$$M^\star = C_m^\star(J)$$

Si $M \geq 1,5 \cdot A$ alors $M^\star \ = \ M$

Algorithm 4: COMBINE

Data: instance de $\mathrm{P_m||C_{max}}$, avec m machines, n jobs, et un coefficient α $(0,005)$

A $= \sum_{i=1}^{n} (\frac{P_i}{m})$

M $\leftarrow C_m^{lpt}(J)$

if $M \geq 1,5 \cdot A$ **then**

  $M^\star \ = \ M$

else

  $C_u \ \leftarrow \ M$

  $C_l \ \leftarrow \ max\{(\frac{M}{\frac{4}{3} - \frac{1}{3 \cdot m}}), P1, 1\}$

  while $C_u - C_l > \alpha \cdot A$ **do**

    appliquer MULTIFIT

  `// on arrête lorsque` $C_u - C_l \leq \alpha \cdot A$

Conplexité	$O(nlogn + knlogm)$
Ratio [7]	$\Gamma(COMBINE) \leq \frac{13}{12} + 2^{-k}$

avec k le nombre d'itérations pour la recherche dichotomique. Concernant la complexité, pour atteindre $C_u - C_l \leq \alpha \cdot A$, généralement, 6 itérations suffisent ($k = 6$). Mais COMBINE a déjà exécuté une fois LPT (k=7).

LISTFIT

Gupta *et al.*, [7], ont aussi l'idée d'utiliser MULTIFIT (3), afin de réaliser l'olgorithme LISTFIT.

Celui-ci sépare la liste des travaux en 2 sous-listes, traitée soit dans un ordre LPT (Longest Time Processing), soit dans un ordre SPT (Shortest Time Processing). Puis LISTFIT combine ces 2 sous-listes en appliquant MULTIFIT à chaque itération.

Algorithm 5: LISTFIT

Data: n, m, p_i for $i = 1, ..., n$

1 let $r = 1$, $q = 1$, and $C_{max} = C_{max}(LPT)$, the makespan obtained by the LPT algorithm. **Goto step 2**.

2 let $\Phi = \varnothing$, A = $\{1, ..., n\}$, and $B = \varnothing$. let ω_r be the sequence of jobs in job-list A sorted according to ordering τ. **Goto step 3**.

3 let $\alpha = C_{max}(MULTIFIT)$ be the makespan obtained by using algorithm MULTIFIT, with $\sigma = \Phi_q \cdot \omega_r$ in step 1 of algorithm MULTIFIT. If $C_{max} > \alpha$ then set $C_{max} = \alpha$ and $\gamma_h = \pi_h$ for $h = 1, 2, ..., m$. If $A \neq \varnothing$ then **goto step 4**; otherwise **goto step 5**.

4 remove the last job of ω_r and place it into B. Update A, Φ_q and ω_r. Let $\sigma = \Phi_q \cdot \omega_r$. **goto step 3**.

5 If $\tau < 2$ then set $\tau = \tau + 1$ and **goto step 2**; otherwise **goto step 6**

6 If $q < 2$ then set $q = q + 1$, $\tau = 1$, and **goto step 2**; oterwise **stop**;
// The schedule where jobs in γ_h are proceded on machine h is an approximate solution of the P||C_{max} problem with makespan C_{max}.

Conplexité	$O(n^2 log(n) + k \cdot n^2 log(m))$
Ratio [7]	$\Gamma(LISTFIT) \leq \frac{13}{12} + 2^{-k}$

avec k le nombre d'itérations pour la recherche dichotomique.

3.2.3 Approche gloutonne

SLACK (Croce *et al.*, 2018) .

Croce *et al.* [3], en effectuant la preuve d'une borne d'approximation pour le développement de LPT-Rev (2), 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 :

$$\begin{array}{ll} \textit{Nombre de jobs} & n = 2 \cdot m + 1 \\ \textit{Avec} & P_{2 \cdot m+1} \geq P_1 - P_m \end{array}$$

Où ils ont planifié d'abord, le job $2 \cdot m + 1$, puis un sous-ensemble de jobs triés $\{1, ..., m\}$ et pour finir un sous-ensemble de jobs triés $\{m + 1, ..., 2 \cdot m\}$

En résulte l'algorithme suivant :

Algorithm 6: SLACK

1 trier la liste des jobs dans l'ordre décroissant des temps nécessaires de traitements

2 réindexer les jobs, de manière à obtenir $P_1 \geq P_2 \geq ... \geq P_n$

3 Découper l'ensemble obtenu en $\frac{n}{m}$ 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 (SLACK) entre le premier job du tuple et le dernier.

$$\begin{array}{cc} \{\{1, ..., m\} & \{m + 1, ..., 2 \cdot m\} ...\} \\ P_1 - P_m & P_{m+1} - P_{2 \cdot m} ... \end{array}$$

5 trier les tuples par ordre décroissant de "Slack" et ainsi former un nouvel ensemble `// e.g :` $\{\{m + 1, ..., 2 \cdot m\}\{1, ..., m\}\}$ `si` $P_{m+1} - P_{2 \cdot m} > P_1 - P_m$.

6 applique l'ordonnancement (Affectation à la machine la moins chargée à ce moment là) à l'ensemble ainsi obtenu.

3.3 Programmation linéaire

L'ordonnancement, et plus particulièrement $\mathrm{P_m}||\mathrm{C_{max}}$ s'inscrit parfaitement dans l'énoncé d'un problème de programmation linéaire. En effet, la fonction objectif i.e minimiser le makespan, ainsi que les contraintes sont des fonctions linéaires. Toutefois, les variables, et le résultat attendu 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.

PA (Mokotoff)

Mokotoff [11] 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 :
Minimiser y tel que :
- $\sum_{j=1}^{m} x_{ij} = 1 \quad$ pour $1 \leq i \leq n$
  Sur toutes les machines, au moins un et un seul x_i est égal à 1.
  Un job est affecté à une, et une seule machine.
- $y - \sum_{i=1}^{n} P_i \cdot x_{ij} \geq 0 \quad$ pour $1 \leq j \leq m$
  Pour une machine donnée, la somme des temps est $\leq$ à y.

Où la valeur optimale de y est C_{max}
et x_{ij} =
- 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
- $n \cdot m + 1$ variables (les variables x_{ij} et la variable y)
- $n + m$ contraintes

La zone F peut être définie ainsi :
$$F = \{(x, y) : x \in B^{n \cdot m}, y \in \mathbb{R}_+ : \textstyle\sum_{j=1}^{m} x_{ij} = 1 \ \forall i; y - \sum_{i=1}^{n} P_i \cdot x_{ij} \geq 0 \ \forall j\}$$
avec $B = \begin{bmatrix} x_{11} & x_{\ldots 1} & x_{n1} \\ x_{1\ldots} & \ddots & x_{n\ldots} \\ x_{1m} & x_{\ldots m} & x_{nm1} \end{bmatrix}$

le polytope P, relatif à F est défini ainsi :
$$F = \{(x, y) : x \in \mathbb{R}_+{}^{n \cdot m}, y \in \mathbb{R}_+ : \textstyle\sum_{j=1}^{m} x_{ij} = 1 \ \forall i; y - \sum_{i=1}^{n} P_i \cdot x_{ij} \geq 0 \ \forall j\}$$

il est possible de construire un ensemble fini d'**inégalités**
$Ax + Dy \leq \bar{b}$ telles que
$min\{y : (x, y) \in F\} \;=\; min\{y : x \in {\mathbb{R}_+}^{n \cdot m}, y \in \mathbb{R}_+ Ax + Dy \leq \underline{b}$
NB : Une solution $(x^\circ, y^\circ) \in P$ doit être exclue (car n'est pas un vecteur entier) si $(x^\circ, y^\circ) \notin P$

Des **inégalités transitoires** peuvent être générées (nombre maxi de jobs par machine)
$\sum_{i \in S_j} x_{ij} \leq L_j \quad (L_j = h - 1 \iff S_{J_h} > LB \; et \; S_{J_{(h-1)}} \leq LB)$
LB : Borne inférieure.

Pour un problème $\mathrm{P_m}||\mathrm{C_{max}}$, même de taille modeste, le nombre de variables et contrites est très important, dont certaines sont inutiles. L'algorithme va donc utiliser la méthode des plans sécants (Cutting Plane Method). À chaque itération , des inégalités valides sont générées, puis une relaxation est exécutée, jusqu'à l'obtention d'une solution faisable.

Algorithm 7: PA

1. Détermination de la borne inférieure (LB) suivant l'algorithme de McNaughton [10].
2. Détermination de la borne Supérieure (UB juste pour la nommer) suivant l'heuristique LPT (1).
3. Si LB coïncide avec UB la solution optimale est trouvée Sinon le processus itératif démarre.
4. À chaque itération un programme de relaxation linéaire est résolu.dans lequel C_{max} 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.
5. Sinon, des nouvelles inégalité (inégalités et/ou inégalités transitoires) sont ajoutées à la nouvelle relaxation linéaire.Le nouveau programme linéaire est résolu et l'algorithme s'arrête si la solution est entière.
6. Si la relaxation n'est pas possible, la limite inférieure (LB) est augmentée d'une unité et le processus itératif redémarre.
7. Par contre, si les inégalités ne peuvent pas être générées, un algorithme *Branch&Bound* prend le relais pour résoudre le problème.

3.4 Approximation

Une catégorie d'algorithmes fournit une garantie d'approche. Notamment les PTAS (Schéma d'Approximation en Temps Polynomial).

PTAS

Un PTAS est un algorithme qui calcule, pour tout $\epsilon > 0$ donné, une solution proche à un facteur $(1 + \epsilon)$ pour un problème de minimisation, ou $(1 - \epsilon$ pour un problème de maximisation, de l'optimal, en temps polynomial dépendant de ϵ.

Hochbaum *et al.* [8] proposent le premier PTAS.

3.5 Autres approches

4 Synthèse

5 Conclusion

Bibliographie

[1] Bo Chen and N Chris. Potts, and gerhard j woeginger. a review of machine scheduling : Complexity, algorithms and approximability. *Handbook of combinatorial optimization*, pages 1493–1641, 1999.

[2] Edward G Coffman, Jr, Michael R Garey, and David S Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1) :1–17, 1978.

[3] Federico Della Croce and Rosario Scatamacchia. The longest processing time rule for identical parallel machines revisited. *Journal of Scheduling*, pages 1–14, 2018.

[4] Michael R Garey and David S Johnson. "strong"np-completeness results : Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3) :499–508, 1978.

[5] MR Garey and DS Johnson. Computers and intractability : A guide to the theory of np-completeness. freeman, san francisco, 1979. 1982.

[6] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2) :416–429, 1969.

[7] Jatinder ND Gupta and Alex J Ruiz-Torres. A listfit heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control*, 12(1) :28–36, 2001.

[8] Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1) :144–162, 1987.

[9] Chung-Yee Lee and J David Massey. Multiprocessor scheduling : combining lpt and multifit. *Discrete applied mathematics*, 20(3) :233–242, 1988.

[10] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1) :1–12, 1959.

[11] Ethel Mokoto. Scheduling to minimize the makespan on identical parallel machines : an lp-based algorithm. *Investigacion Operative*, 97107, 1999.

[12] Michael H Rothkopf. Scheduling independent tasks on parallel processors. *Management Science*, 12(5) :437–447, 1966.