Historique des travaux autour du problème $P||C_{\max}$

florian colas

14 décembre 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, capables de communiquer et de coopérer

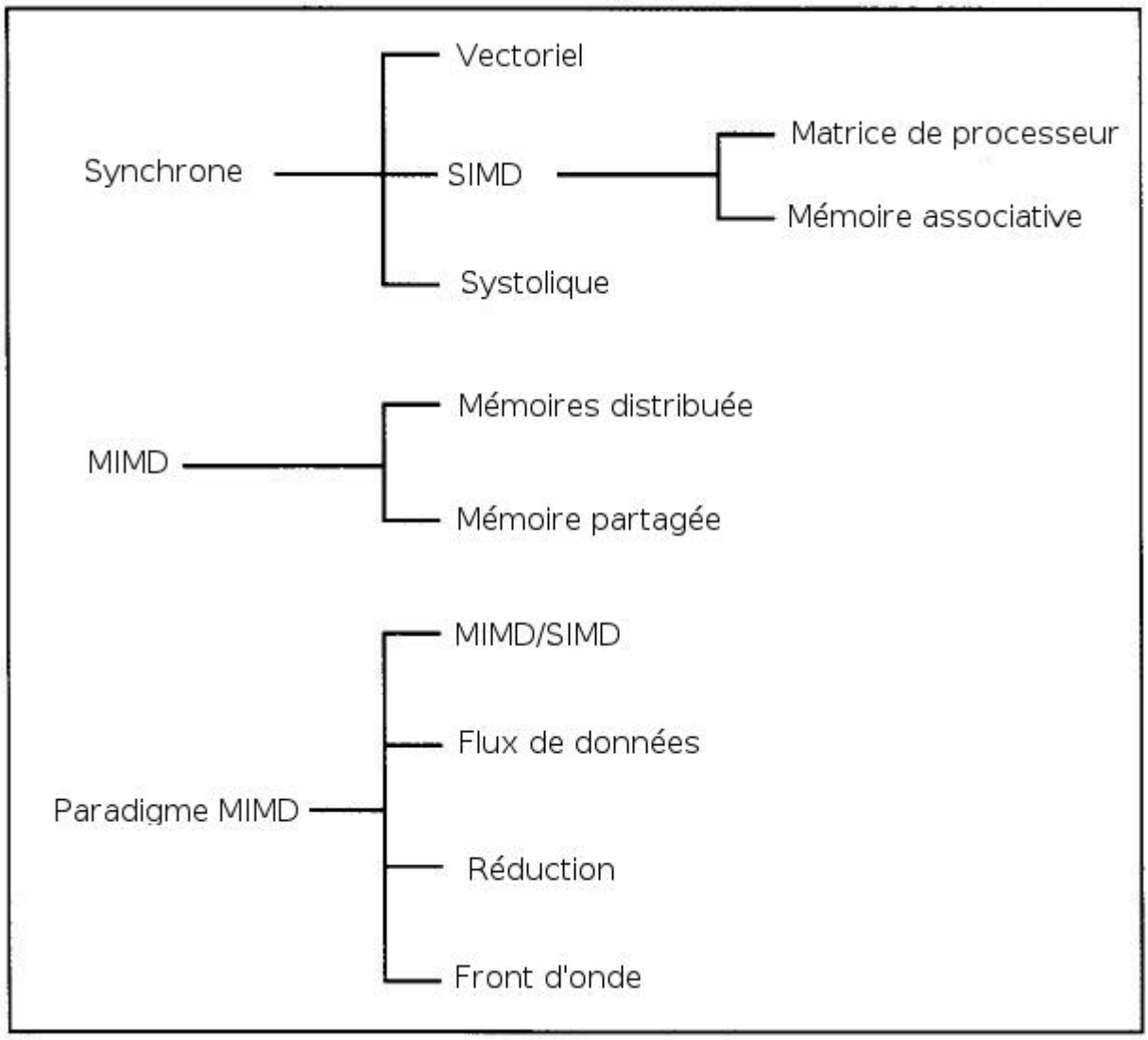
En 1972, Flynn propose une première classification des architectures parallèles [5]. Cette taxonomie repose sur le nombre de flux de données (simple ou multiple) et le nombre de flux d'instructions (simple ou multiple).

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-ordinateurs) Multi processeurs à mémoire partagée (multi-cœurs)

Taxonomie de Flynn

En 1988, Skilicorn étoffe la taxonomie de Flynn et propose une classification plus complète [16], basée sur l'architecture, avec 28 taxons.

En 1990, Duncan présente une taxonomie de haut niveau [4], reprenant les bases de la classification de Flynn, tout en incluant les architectures modernes, et celles qui n'ont pas encore été considérées comme architectures parallèle (e.g es processeurs vectoriels à chaînes de traitement).



Taxonomie de haut niveau de l'architecture de l'informatique parallèle.

— Les architecture synchrones. Ces architectures coordonnent les opérations concurrentes via une horloge globale et un contrôleur de programmes.(i) Les processeurs vectoriels permettent un traitement parallèle en envoyant séquentiellement des éléments vectoriels dans des pipelines d'unités de traitements mis en cascades.(ii) Les machines systoliques, pulsent le déplacement des données entre mémoire et réseau de processeurs,à un rythme soutenu. (iii) Les SIMD, emploient une unité de contrôle centrale, et de multiples processeurs. On distingue les SIMD à matrice de processeurs, (calibrés pour le calcul scientifique à grande échelle, le traitement d'images) où on retrouve

les GPU, et les SIMD à mémoire associative (particulièrement adaptés pour le traitement de bases de données, le suivi, la surveillance).

— Les architectures asynchrones, de type MIMD, emploient plusieurs processeurs pour exécuter des flux d'instructions indépendant, en utilisant des données locales. On distingue, d'une part, les MIMD à mémoire distribuée. Les processeurs ont leur propre mémoire, et le partage d'information s'opère uniquement par échange de messages. On rencontre dans ce type d'architecture, les grappes, les MPP (Massive Parallel Processing) et les COW (Cluster Of Workstations). Et d'autre part les MIMD à mémoire partagées, telles que les machines muti-processeurs, ou les processeurs multi-cœurs.

— Les architectures mixtes, ou basées sur le paradigme MIMS. (i) Les hybrides MIMS/SIMS sont des architectures MIMD pilotées de la même façon que les SIMD. (ii) Les architectures à flots de données sont pilotées par les dépendances des données elles-mêmes. (iii) Les architectures à réductions sont utilisée pour les langages de programmation fonctionnels, dont les expressions sont récursives. (iv) Les architectures Front d'onde (wavefront) sont un mélange des architectures à flots de données et des architectures systoliques.

LP : la classification de Flynn date un peu... Peut-être faut-il trouver qlqchose de plus récent

FCO : ok, fait

Les premières machines parallèles étaient des réseaux d'ordinateurs, et des machines vectorielles (faiblement parallèles, très coûteuses), telles que l'IBM 360, les Cray1.

LP : Ce n'est pas si simple. On continue de faire, et d'utiliser des machines vectorielles, les GPU sont aussi du SIMD...

FCO : ok, fait

Le Supercomputer Fugaku (Fujitsu), du Centre pour la science des ordinateurs de RIKEN au Japon, vient de détrôner (www.top500.org parution de juin 2020) le Summit (IBM)de l'Oak Ridge National Laboratory aux États-Unis. Il compte 7,630,848 cœurs basés sur des processeurs ARM A64FX 48C cadencés à 2.2GHz, pour 5,087,232 GB de ram. Il fonctionne sous RedHat.

LP : pense à compléter cette partie avec des infos sur des machines actuelles. voir top500.org dont il serait peut-être plus judicieux de citer les infos : puissance des plus gros ordinateurs, nombre de proc, cores, etc. Parler peut-être aussi des clouds ? Des applications ? Pour intro

FCO : ok, fait

On peut définir une machine parallèle comme un ensemble de processeurs capables de coopérer et de communiquer dans le but de résoudre un problème plus rapidement. Les **clusters**, ou grappes (MIMD à mémoire distribuée), sont principalement homogènes. Avec internet, il devient possible d'interconnecter des ordinateurs, voire des grappes, et de créer ainsi, à l'instar d'un réseau de distribution d'électricité, un système distribué grande échelle, appelé **grid**, ou grille. Une grille relie des **ordinateurs** (PC, portable, Superordinateur, cluster), des **systèmes de bases de données** (e.g base du génome humain, atlas astronomique de Strasbourg), et dans certains cas, des **instruments spéciaux** (e.g le radio-télescope d'Arecibo). Les applications liées aux grilles sont diverses. On peut citer le web (la première grille), le *"Cloud computing"*, les web-services, les réseaux *"peer-to-peer"*, l'*internet computing*, comme le projet SETI (www.boinc-af.org/setihome.html). Ce projet consiste à rechercher des preuves possibles de transmissions radio provenant d'intelligences, autres que terrestres, à l'aide de données fournies par un radiotélescope. Le "serveur", distribue, une portion d'échantillons radio à traiter aux

clients volontaires (logiciel client installé sur un PC), pour ensuite recouper les résultats. En tout, ce sont cinq millions de participants, qui ont accumulé deux millions d'années de calculs depuis la création du projet en 1999. Se pose alors le problème de l'attribution des tâches aux postes clients, un problème d'ordonnancement.

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, 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, et est un champ de la recherche opérationnelle, très actif depuis plus d'un siècle.

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 q_i 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 tâches 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.

LP : définir la notation de Graham à la fin de cette partie puisque tout y est déjà dit.

FCO : ok, fait

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 $P||C_{\max}$se définit alors ainsi :

— $\alpha = \alpha 1\ \alpha 2$, 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 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. $\gamma = \mathrm{C}_{\max}$: on cherche à optimiser le makespan, i.e le temps de traitement total.

2.3 Énoncé du $P||C_{\max}$

Definition 1. $P||C_{\max}$

$P||C_{\max}$consiste à planifier un ensemble $J = \{j_1, j_2, \ldots, j_n)$ 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 $i \in \mathbb{N}$) 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.

la notation :

— $P||C_{\max}$ ou $P_m||C_{\max}$ précise que le nombre de machines m est fixe, et est connu à l'avance dans l'instance du problème, mais reste une variable.

— $P_2||C_{\max}$fixe le nombre de machines parallèles. ici, deux.

LP : on peut recontextualiser l'intérêt du problème. A l'heure actuelle, bon nombre de centres de calcul ont un parc assez homogène. De même les clouds offrent des instances de VM, ce qui permet d'avoir un environnement d'exec homogène

LP : faire un dessin illustratif ?

LP : dire que la difficulté repose uniquement sur le regourpement des jobs sur une machine puisque les machines sont identiques

FCO : Rechercher exemples

2.4 Problématique

Comme l'ont démontré Garey et Johnson, $P_2||C_{\max}$est un problème NP-Difficile [6], et $P||C_{\max}$est un problème NP-Difficile au sens fort [7]. Cependant, $P||C_{\max}$devient un problème NP-Difficile, du moment que le nombre de machines est fixé [1], comme l'a montré Rothkopf [15], qui a présenté un algorithme de programmation dynamique.

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

LP : approximation ?

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

FCO : c'est bien le bon terme

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) $J = \{J_1, J_2, ..., J_n\}$ dont chaque job J_j a un temps de traitement connu $P_j \in P = \{P_1, P_2, ..., P_n\}$
- un ensemble de m machines parallèles identiques $M = \{M_1, M_2, \ldots, M_m\}$

LP : au besoin définir $M = \{m_1, m_2, \ldots, m_m\}$ mais du coup il faut choiir à quoi sert le m

FCO : M (majuscule) est l'ensemble des m machines parallèles identiques. m_i est une machine de M, avec $1 <= i <= m$, il y a double emploi entre m la machine et m le nombre de machines, mais m en tant que nombre est utilisé (aussi) dans $P_m||C_{\max}$. Donc j'ai tout mis en majuscule lorsqu'il s'agit d'une entité, (machine, job, temps) et en minuscule, pour les nombres

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

LP : à quoi sert j ici ?

FCO ok, fait : C'était une erreur de frappe ...le copier/coller qui va avec

— $C_j^\star(J)$ Le makespan optimal, idéal, de l'ordonnancement d'un ensemble J de jobs, sur m machines parallèles identiques.

LP : idem

FCO : ok, fait

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

LP : idem

FCO : ok, fait

3.2 Heuristiques

LP : définir heuristic ?

FCO : ok, fait

Le but d'une heuristique (du grec *heuriskein* : trouver) est de trouver une solution respectant les contraintes du problème, et de bonne qualité selon le critère d'optimisation considéré. La solution ne sera pas forcément optimale, mais une heuristique efficace tente de trouver une solution de bonne qualité, suivant le temps de résolution imparti. e.g pour le voyageur de commerce, on peut choisir l'heuristique "du plus proche voisin". on choisit la ville la plus proche de la ville courante, jusqu'à avoir visité toutes les villes. Cette heuristique est simple, mais donne un très mauvais résultat.

Ces algorithmes donnent des résultats, dont la borne minimale $max_j\{P_j\} \leq C_m^A(J)_{minimal} \leq \dfrac{\sum_{j=1}^{n} P_j}{m}$ et la borne maximale (au pire des cas $C_m^A(J)_{maximal}$

FCO Trouver référence dans les publis

n'est pas maîtrise. une étude du comportement est nécessaire pour définir cette borne maximum.

Les heuristiques représentent la plus grande partie des recherches concernant le problème d'ordonnancement.

LP : on a aussi une borne min qui est le max entre la plus grande des tâches et sum(pi)/m

FCO : ok, fait

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. Le premier job de la liste étant affecté à la première machine disponible.

LP : un algorithme de liste affecte aussi un job, le premier de la lsite, à la première machine prête

FCO : ok, fait

LPT rule (Graham *et al.*, 1969) Graham propose [8] ***Longest Processing Time (LPT) rule***.

Algorithm 1: LPT Rule

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

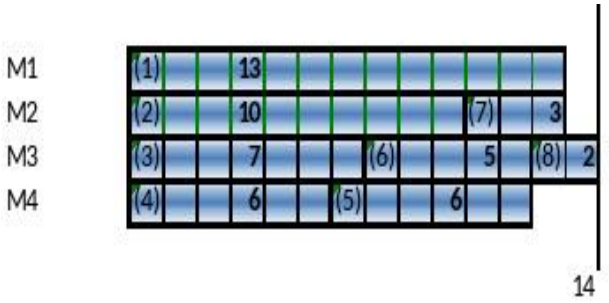
1 Trie les jobs de l'ensemble J dans l'ordre décroissant de leur temps d'exécution et ré-indexe l'ensemble de telle manière à obtenir : $P_1 \geq P_2 \geq \ldots \geq P_n$

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

Exemple

FCO : Algo en anglais

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

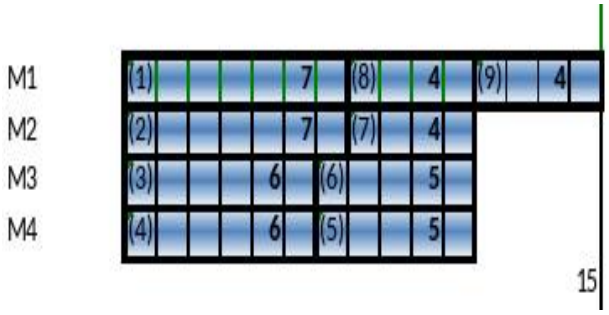
Le tri puis l'affectation s'effectuent en	$O(n \log n + n \log m)$
Le ratio d'approximation	$\Gamma(LPT) \leq \frac{4}{3} - \frac{1}{3m}$

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

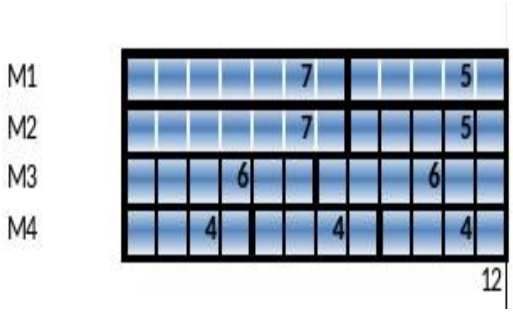
Soit $P = \{7, 7, 6, 6, 5, 5, 4, 4, 4\}$, l'ensemble des P_j déjà triés dans l'ordre décroissant à appliquer sur 4 machines parallèles identiques :

LPT donne le résultat suivant



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

Un ordonnancement optimal aurait été :



$C_4^\star(J) = 12$
Soit une marge 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.

Definition 2. Job et machine critique
Le job critique (noté J') est le job qui détermine le makespan.
La machine critique est la machine qui exécute le job critique.

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.*

Le rapport d' $\frac{4}{3} - \frac{1}{3(m-1)}$ est inférieur au ratio $\frac{4}{3} - \frac{1}{3m}$ (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 $P||C_{\max}$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.

FCO Algo en anglais

Algorithm 2: LPT-Rev

Data: instance de $P||C_{\max}$, 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** $min(z_1, z_2)$

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 $P||C_{\max}$. Il consiste à ranger des objets de taille différentes, dans des bacs identiques, tout en minimisant leur nombre.

L'ensemble des n jobs $J = \{J_1, J_2, \ldots, J_n\}$, et de leurs temps de traitement $P_j \in P = \{P_1, P_2, ..., P_n\}$, peuvent être vus respectivement comme :
— un ensemble d'objets $T_j \in T = \{T_1, T_2, ..., T_n\}$
— leur taille $L(T_j)$ *avec* $1 \leq j \leq n$
Une taille maximale C des bacs (ou boites) est donnée.

Definition 3. Packing

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

le problème Bin-packing est NP-Complet [11].

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

LP : petite remarque sur la complexité du problème ?

FCO : ok, fait

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.

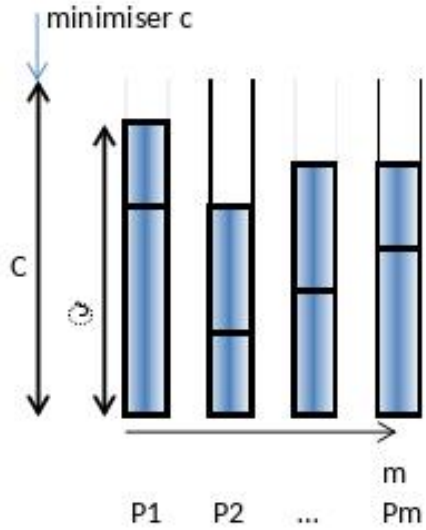
Soit $T^{\star}_m = min\{C : FFD(T, C) \leq m\}$ la plus petite valeur de C (taille des bacs) qui permet à T d'être pacqué dans m (ou moins) bacs.

L'idée de MULTIFIT est donc de proposer une valeur pour C, faire tourner $FFD(T, C)$, et réduire à chaque itération cette valeur de C jusqu'à ce que le nombre m de bacs renvoyé par $FFD(T, C)$, alors devenu insuffisant, augmente à $m + 1$. Cette valeur charnière de C est $T^{\star}_m$, qui correspond au makespan minimum recherché, de l'ordonnancement de l'ensemble T de jobs sur m machines identiques parallèles. La difficulté, est de proposer une première valeur de C pas trop éloignée de la valeur recherchée, afin de réduire au maximum le nombre d'itérations.

LP : deux mots pour dire comment est implémentée la recherche

FCO : ok, fait

La méthode utilisée est une recherche dichotomique. Après avoir défini les bornes supérieure $C_u = \max\{\frac{2}{m} \cdot L(T), \max_i\{L(T_i)\}\}$ et inférieure $C_l = \max\{\frac{2}{m} \cdot L(T), \max_i\{L(T_i)\}\}$ de départ, FFD est exécuté avec le milieu $C_d = \frac{C_u + C_l}{2}$. si $FFD(T, C_d) \leq m$ alors la nouvelle fourchette de recherche devient $C_u = C_d$ et C_l, sinon (si $FFD(T, C_d) > m$) alors la nouvelle fourchette devient C_u et $C_l = C_d$. Un nombre d'itérations k et donné à l'exécution de MULTIFIT, estimé en fonction de la taille du problème (n et m). Mais même pour un problèmes de grande taille, le résultat ne change plus après 7 itérations. Il est donc inutile d'utiliser $k > 7$.



Fonctionnement de FFD et principe de MULTIFIT

Algorithm 3: MULTIFIT

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

1 La recherche de $T^\star_m$ 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 $FFD[T,C] \leq m$

FCO : algo en anglais + enlever les goto

LP : k fixé ? ou seuil entre deux iter ?

FCO : ok, fait

Tri puis k FFD s'effectuent en	$O(n \log n + kn \log m)$
Ratio [12]	$\Gamma(MULTIFIT) \leq 1,220 + 2^{-k}$

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

COMBINE Lee *et al.*, [12] ont l'idée d'utiliser LPT (1) pour réduire les bornes de départ et ainsi optimiser MULTIFIT (3) dans un algorithme nommé COMBINE.

LP : développer/préciser un peu

FCO : ok, fait

soient

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

$$\textit{et} \qquad M = C_m^{lpt}(J)$$

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

COMBINE, améliore certains point de l'algorithme MULTIFIT en se basant sur les principes suivants :

- Le résultat M obtenu par LPT, est forcément supérieur ou égal à $M^\star$, soit $M > M^\star$. COMBINE utilise donc $M = C_m^{lpt}(J)$ comme borne supérieur C_u de départ pour la recherche dichotomique utilisée par MULTIFIT. celle-ci est beaucoup plus proche du résultat optimal que la borne supérieure de départ $2 \cdot A$ utilisé à l'origine par MULTIFIT.
- Si Le résultat M obtenu par LPT est tel que $M \geq 1,5 \cdot A$ alors $M = M^\star$. Il n'est donc pas nécessaire de poursuivre la recherche.
- Au lieu de prédéterminer un nombre d'itérations, COMBINE utilise une condition d'arrêt de la recherche dichotomique, en fonction de l'écart entre les deux bornes. COMBINE stope la recherche dès que $C_u - C_l \leq \alpha \cdot A$. Les tests de COMBINE ont été effetués avec $\alpha = 0,005$

Algorithm 4: COMBINE

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

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

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

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

4 $\quad M^\star = M$

5 **else**

6 $\quad C_u \leftarrow M$

7 $\quad C_l \leftarrow \max\{(\frac{M}{\frac{4}{3} - \frac{1}{3 \cdot m}}), P1, 1\}$

8 $\quad$ **while** $C_u - C_l > \alpha \cdot A$ **do**

9 $\quad\quad$ appliquer MULTIFIT

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

FCO Algo en anglais

Conplexité	$O(n \log n + kn \log m)$
Ratio [9]	$\Gamma(COMBINE) \leq \frac{13}{12} + 2^{-k}$

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.*, [9], 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 $i = 1, \ldots, n$

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 $C_{\max}$.

FCO enlever les goto

Conplexité	$O(n^2 \log(n) + k \cdot n^2 \log(m))$
Ratio [9]	$\Gamma(LISTFIT) \leq \frac{13}{12} + 2^{-k}$

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.

LP : est-ce qu'on sait dire pourquoi ? faire une petite analyse

notamment pour l'instance suivante :

Nombre de jobs	$n = 2 \cdot m + 1$
Avec	$P_{2 \cdot m+1} \geq P_1 - P_m$

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.

$$\{\{1, ..., m\} \qquad \{m + 1, ..., 2 \cdot m\}...\}$$
$$P_1 - P_m \qquad P_{m+1} - P_{2 \cdot m}...$$

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 $P||C_{\max}$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 [14] présente un algorithme basé sur la formulation de la programmation linéaire, en utilisant des variables booléennes d'affectation des jobs à une machine.

LP : définir x_{ij}

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

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

il est possible de construire un ensemble fini d'**inégalités**
$Ax + Dy \leq \overline{b}$ 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)
$\sum_{i \in S_j} x_{ij} \leq L_j \quad (L_j = h - 1 \iff S_{J_h} > LBetS_{J_{(h-1)}} \leq LB)$
LB : Borne inférieure.

Pour un problème $P||C_{\max}$, même de taille modeste, le nombre de variables et contraintes 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 [13].
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).

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.

PTAS (Hochbaum *et al.*) Hochbaum *et al.* [10] proposent le premier PTAS adapté au problème $P||C_{\max}$. Cette approche s'appuie aussi sur le problème semblable à $P||C_{\max}$, celui du bin packing et prend MULTIFIT (3) comme réflexion de départ. Le problèmebin-packing est posé différemment :

Soient :

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

— leur taille $L(T_j)$ *avec* $1 \leq j \leq n$ et $0 \leq L(T_j) \leq 1$

Une taille maximale C des bacs (ou boites).

$T^\star(T)$, définit plus haut, correspond au bin-packing idéal, et donc, au nombre minimum de bacs nécessaires pour l'organisation des pièces de l'en-

semble T. Le but de l'algorithme proposé, et de construire un arrangement bin-packing, qui utilise au plus $T_m^\star$ bacs, remplit avec des pièces totalisant une taille au plus de $1+\epsilon$, et ainsi donner une réponse au problème $P||C_{\max}$en temps polynomial.

Le problème bin-packing et $P||C_{\max}$sont reliés de la façon suivante : $T^\star(\frac{J}{C}) \leq m$ si et seulement si $C_m^\star(J) \leq C$. Donc la taille minimale des bacs, $C^\star$ correspondant au délai idéal makespan, est telle que $T^\star(\frac{J}{C^\star}) \leq m$.

Nous sommes en présence de deux paramètres critiques :

— m le nombre de machines parallèles identiques ;

— C la taille des bacs, et aussi, le makespan recherché.

Il y a donc Deux problèmes d'optimisation, dont l'un consiste à fixer le premier paramètre pour optimiser l'autre, et inversement, fixer l'autre pour optimiser le premier. Ce qui constitue un problème de reconnaissance (ou décision) à deux paramètres $R(p_1, p_2)$.

Problème primal et approximation $\epsilon-$primal

soit le problème primal $(J, \bar{p_1})$ où p_1 est donné en paramètre et p_2 est optimisé.

La valeur optimale est $OPT_P(J, \bar{p_1})$.

soit l'algorithme d'approximation $\epsilon-$primal pour le problème primal, qui donne comme résultats :

— poru le premier paramètre p_1 une valeur $\leq \bar{p_1}$

— pour le deuxième paramètre p_2 une valeur $PRIMAL_{P,\epsilon} \leq (1+\epsilon) \cdot OPT_P(J, \bar{p_1})$

Problème dual et approximation $\epsilon-$dual

soit le problème primal $(J, \bar{p_2})$ où p_2 est donné en paramètre et p_1 est optimisé.

La valeur optimale est $OPT_D(J, \bar{p_2})$.

soit l'algorithme d'approximation $\epsilon-$dual pour le problème dual, qui donne comme résultats :

— pour le premier paramètre p_1 une valeur $DUAL_{D,\epsilon}(J, \bar{p_2}) \leq OPT_D(J, \bar{p_2})$.

— pour le deuxième paramètre p_2 une valeur $\leq (1+\epsilon) \cdot \bar{p_2}$

Trouver un algorithme $\epsilon-$primal pour le problème primal, peut être réduit à trouver un algorithme $\epsilon-$dual pour le problème dual.

Trouver un algorithme d'approximation $\epsilon-$dual, peut être réduit à trouver un algorithme d'approximation $\epsilon-$dual pour une instance du problème où la taille des pièces sont $> \epsilon$.

L'intervalle de la taille des pièces (devenu $]\epsilon, 1]$) est découpé en S = $\lceil \frac{1}{\epsilon^2} \rceil$ sous intervalles de taille identique $]\epsilon = l_1, l_2],]l_2, l_3], \ldots,]l_S, L_{S+1} = 1]$. Chaque bac P_i, sera rempli au maximum avec $\lfloor \frac{1}{\epsilon} \rfloor$ pièces.

Soit $x_k avec 1 \leq k \leq S$ le nombre de pièces d'un bac dont la taille $\in]l_k, l_{k+1}]$. Chaque x_k peut prendre une valeur dans l'intervalle $[0, \frac{1}{\epsilon \acute{L}}[$

La configuration d'un bac P_i peut être donnée pas un s-tuple $(x_1, x_2, \ldots, x_S)$. Une configuration est dite faisable si $\sum_{k=1}^{S} x_k \cdot l_k \leq 1$ avec l_k de $]l_k, l_{k+1}]$.

Nous avons donc
$L(P_i) \leq \sum_{k=1}^{S} x_k \cdot l_{k+1}$
$\leq \sum_{k=1}^{S} x_k \cdot (l_k + \epsilon^2)$
$= \sum_{k=1}^{S} x_k \cdot l_k + \epsilon^2 \cdot \sum_{k=1}^{S} x_k$
$\leq 1 + \epsilon^2 \cdot \frac{1}{\epsilon}$
$= 1 + \epsilon$

Soit $b_k avec 1 \leq k \leq S$ le nombre de pièces utilisées dans tous les bacs dont la taille $\in]l_k, l_{k+1}]$.

Soit $Bins(b_1, b_2, \ldots, b_S)$ le nombre minimum de bacs nécessaires, lorsqu'il y a b_k pièces de taille l_k, et considérons que le premier bac soit rempli, nous obtenons :

$Bins(b_1, b_2, \ldots, b_S) = 1 + \min_{configuraion faisable(x_1, \ldots, x_S)} Bins(b_1 - x_1, b_2 - x_2, \ldots, b_S - x_S)$

Ce qui est résolu par programmation dynamique.

FCO Algos $\epsilon = 1/5$ et $1/6$

3.5 Autres approches

3.5.1 Partitionnement

LDM

FCO Rappels, Lipshitz, fonction densité, loi triangulaire, loi beta

3.5.2 autre

algorithmes génétiques

algorithmes à réseaux de neurones

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] R. Duncan. A survey of parallel computer architectures. *Computer*, 23(2) :5–16, 1990.

[5] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9) :948–960, 1972.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3 :299–325, 1974.

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

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

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

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

[16] D. B. Skillicorn. A taxonomy for computer architectures. *Computer*, 21(11) :46–57, 1988.