UNIVERSITÉ DE NANTES

Distanciel – Complexité et algorithmes

Min Makespan

Sommaire

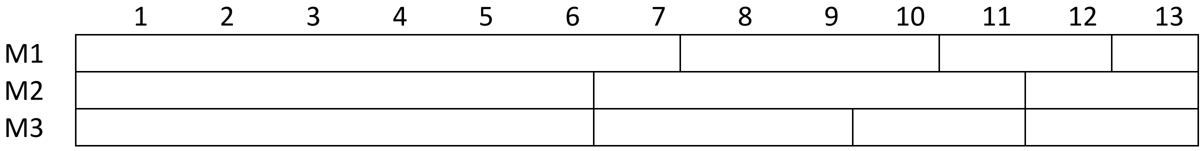
Marc HARTLEY – Maxime VENTE

Préambule - Réponses aux exercices

1. **Indiquer ce que donne l'algorithme LPT sur l'exemple de l'exercice 3.2 de la feuille de TD3, sous la forme d'un dessin similaire à la Figure 1 de l'exercice 3.2.**

Pour rappel, les valeurs de l'instance de l'exercice 3.2, une fois triés dans l'ordre décroissant, sont :

T = 7 ; 6 ; 6 ; 5 ; 3 ; 3 ; 2 ; 2 ; 2 ; 1 et N = 11 et M = 3



On retrouve donc T_{LPT} = 13

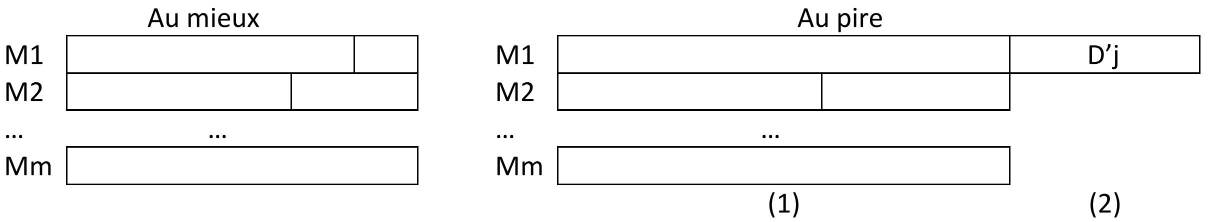
2. **Quel est le ratio d'approximation obtenu par LPT sur cet exemple ?**

On sait que T_{opt} = 13 ($T_{opt\_borne\_max}$ = 7 et $T_{opt\_borne\_moyenne}$ = 39 / 3 = 13)

Donc le ratio $\frac{T_{opt}}{T_{LPT}} = \frac{13}{13} = 1.$

3. **Montrer que pour toute instance I, $T_{LPT} \leq \frac{\sum_{k\neq j} d'_k}{m} + d'_j$, où j est le numero de la tache qui termine en dernier.**

Voyons les cas au mieux est au pire de LPT :



On a en effet le pire scenario dans lequel toutes les taches se terminent en même temps, sauf la dernière, qui commence à la terminaison des autres.

On a alors T_{LPT} au pire qui est égal à (1) + (2). Or $(1) = \frac{\sum_{k\neq j} d'_k}{m}$ et $(2) = d'_j$, donc $T_{LPT} \leq \frac{\sum_{k\neq j} d'_k}{m} + d'_j$.

4. **Supposons pour commencer que $n \leq m + 1$. En déduire dans ce cas que LPT est toujours optimal.**

Si $n \leq m$, une seule tache au maximum n'est affectée à chaque machine. Donc $T_{LPT} = max(d)$ (T_{LPT} est la durée maximale d'une tache).

Or on sait que $T_{opt} \geq max(d)$, on peut alors en conclure que $T_{LPT} = T_{opt} = max(d)$ et donc que T_{LPT} est optimal pour $n \leq m$.

5. **Montrer que $T_{opt}(I) \geq 2d'_{m+1}$.**

On sait que $d'_k \geq d'_{m+1}$ pour $k \leq m + 1$. Avec $n > m$, chaque machine a au moins une tache de durée $d \geq d'_{m+1}$. (Donc $m_k \geq d'_{m+1}$ avec m_k la durée cumulée des taches sur une machine).

À partir de la tache m+1, au moins une machine a au moins 2 taches de durée $d \geq d'_{m+1}$. Le résultat de LPT ne pouvant qu'augmenter, on en déduit que $T_{LPT}(I) \geq 2 * d'_{m+1}$.

6. **Appelons j le numéro de la tache qui se termine en dernier quand LPT est appliqué. Montrer que l'on est nécessairement dans un de ces deux cas suivants : (a) $T_{LPT}(I) = T_{opt}(I)$ ou (b) $j \geq m + 1$.**

En effet, jusque-là on a déduit pour LPT :

$$\begin{cases} T_{LPT} = T_{opt}\ pour\ n < m+1 \\ T_{LPT} \geq 2 * d'_{m+1}\ pour\ n \geq m+1 \end{cases}$$

Or j est la dernière tache, donc $j = n$:

$$\begin{cases} T_{LPT} = T_{opt}\ pour\ j < m+1 \\ T_{LPT} \geq 2 * d'_{m+1}\ pour\ j \geq m+1 \end{cases}$$

Les 2 cas sont distincts par leur condition sur j, on en conclue que $T_{LPT}(I) = T_{opt}(I)\ ou\ j \geq m + 1$.

7. **En vous appuyant sur les questions précédentes, montrer que pour toute instance I, $T_{LPT}(I) \leq r * T_{opt}(I)$ où r est une constante dont vous donnerez la valeur.**

Nous savons que pour $n \leq m$, $T_{LPT}(I) = T_{opt}(I)$ donc cela ne nous intéresse pas pour trouver r.

Pour $n \geq m + 1$, on sait que

$$\begin{gathered} T_{LPT} \leq \frac{\sum_{k \neq j} d'_k}{m} + d'_j \\ \equiv T_{LPT} \leq \frac{\sum d'_k}{m} - \frac{d'_j}{m} + d'_j \\ \equiv T_{LPT} \leq T_{opt} + \frac{(m+1)}{m} d'_j \\ \equiv T_{LPT} \leq T_{opt} + d'_j \\ \equiv T_{LPT} \leq 2 * T_{opt} \\ \equiv r = 2 \end{gathered}$$

8. **Conclure quant au ratio d'approximation de l'algorithme LPT.**

On en conclue alors que LPT est 2-approximation.

9. **Donner, en la justifiant, la valeur de $T_{opt}(I_m)$. Illustrer votre réponse sur l'instance I_5 (donc m=5).**

On a $T_{opt}(I_m)_{borneMax} = m + m - 1$ et $T_{opt}(I_m)_{borneMoyenne} = \frac{3m + \sum_{i<m} 2m+2i}{m}$

On peut traduire par :

$$T_{opt}(I_m)_{borneMoyenne} = \frac{3m + (m-1) * (2m+m)}{m} = \frac{3m + (m-1) * 3m}{m} = \frac{3m + 3m^2 - 3m}{m} = 3m$$

On a donc $T_{opt}(I_m)_{borneMax} < T_{opt}(I_m)_{borneMoyenne}$, donc $T_{opt}(I_m) \geq 3m$

Par exemple, pour I_5: $T_{opt}(I_5) \geq 3 * 5 \Rightarrow T_{opt}(I_m) \geq 15$

10. Donner, en la justifiant, la valeur de $T_{LPT}(I_m)$. Illustrer votre réponse sur l'instance I_5.

Si on suppose avoir uniquement $2m$ tâches (seulement 2 tâches de durée m), on retrouve obligatoirement un $T_{TLP} = T_{opt}$. En effet, chaque machine a exactement 2 tâches, et chaque tâche a un « opposé » : la dernière tâche sera avec la première, l'avant-dernière avec la seconde, etc… De ce fait, les machines ont toutes un temps d'exécution $d_i + d_{n-i} = [2 * m - (i + 1)] + [m + i] = 3 * m - 1$. Maintenant on rajoute la dernière tâche qui se lance sur M_1 (car toutes les taches se sont finies en même temps), on trouve :

$$T_{LPT}(I_m) = 3 * m - 1 + m$$

$$T_{LPT}(I_m) = 4m - 1$$

Par exemple, pour I_5: $T_{LPT}(I_5) = 4 * 5 - 1 = 19$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
M1	9									5					5					
M2	9									5										
M3	8								6											
M4	8								6											
M5	7							7												

11. Quelle est la valeur du ratio $\frac{T_{LPT}(I_m)}{T_{opt}(I_m)}$ lorsque m devient grand ? En déduire une borne inférieure sur le ratio d'approximation de l'algorithme LPT.

On a donc un ratio $\frac{4m-1}{3m}$, et donc $\lim_{m\to+\infty} \frac{4m-1}{3m} = \frac{4}{3}$.

On peut considérer $\frac{4}{3}$ comme borne inférieur pour LPT.

12. Donner, en la justifiant, la valeur de $T_{LSA}(I_m)$. Illustrer votre réponse sur l'instance I_5.

Les durées sont générées dans un ordre croissant, donc on affecte toutes les tâches de la plus courte à la plus longue, ce qui oblige le fait qu'une machine à laquelle on vient d'affecter une tâche est actuellement la machine la plus longue (et donc la dernière à recevoir une nouvelle tâche). Si on répète cette règle lors de la distribution de tâches, on peut affirmer qu'une machine M_i recevra obligatoirement la tâche i, $i+m$, $i+2*m$, etc… Or on sait qu'il n'y a que $2m+1$ tâches, donc la seule machine à recevoir 3 tâches (dont la dernière) est la machine M_1. On en conclue que :

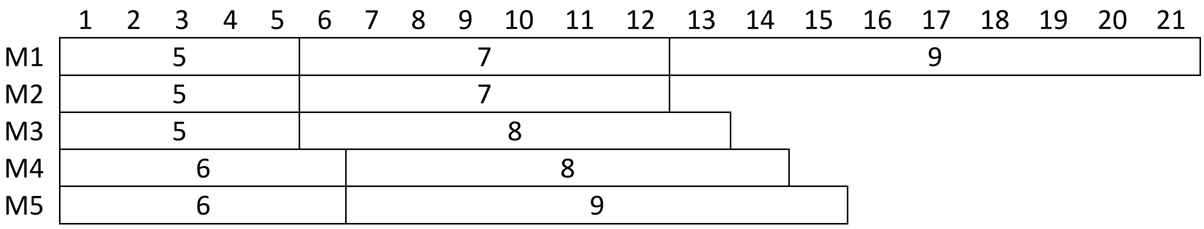
$$T_{LSA}(I_m) = d_1 + d_{1+m} + d_{2m+1}$$

$$T_{LSA}(I_m) = [m] + [m+x] + [m-1]$$

On peut calculer x en sachant que la tâche $m+1$ vaut $division\ euclidienne\left(\frac{m-1}{2}\right)$. On sait que $d_{2m+1} = m-1$. Soit :

$$T_{LSA}(I_m) = 4m + division\ euclidienne\left(\frac{m-3}{2}\right)$$

Par exemple, pour I_5: $T_{LSA}(I_5) = 4*5 + div\left(\frac{2}{2}\right) = 20 + 1 = 21$



13. Quelle est la valeur du ratio $\frac{T_{LSA(I_m)}}{T_{opt}(I_m)}$ lorsque m devient grand ?

On a alors le ratio $\frac{4m+div\left(\frac{m-3}{2}\right)}{3m}$, et donc $\lim_{m\to+\infty} \frac{4m+\frac{m}{2}}{3m} = \frac{\frac{9m}{2}}{3m} = \frac{3}{2}$.

1. Introduction

1.1. Résumé du sujet

Le problème de Makespan (ou « répartition de tâches ») consiste à répartir différentes tâches sur plusieurs machines dans le but de réaliser toutes les tâches dans le plus court intervalle de temps. Chaque machine est considérée identique, aucune ne fonctionne plus vite qu'une autre ou n'est meilleure dans une certaine tâche.

Ce problème est NP-dur, il ne nous est donc impossible de réaliser un algorithme de complexité polynômiale pour un résultat exact (si P ≠ NP). Le but sera d'implémenter certains algorithmes d'approximation, puis de proposer le nôtre.

1.2. Algorithmes

Nous avons implémenté 3 algorithmes :

- LSA (*List Simulated Annealing*)
- LPT (*Longest Processing Time*)
- MyAlgo (*Vente-Hartley*)

Cela nous permettra d'étudier ces algorithmes, leur complexité, leur approximation, …

1.3. Équipe

L'équipe se compose de VENTE Maxime et **HARTLEY Marc.**

2. Le programme

2.1. Conception

Nous souhaitons un programme en console compatible en multi-plateforme, de plus le projet est court et peu de fichiers seront nécessaires. Nous avons donc choisi le langage Java que nous maitrisons tous deux.

2.2. Interface

L'interface se fait dans un terminal.

Un point qu'il nous a semblé important est de permettre à l'utilisateur d'exécuter le programme de la façon qu'il préfère. Il est donc important d'avoir un menu principal pour diriger l'utilisateur, mais aussi la possibilité de lancer le programme avec des options pour avoir directement un résultat sans interaction.

2.3. Structure de données

Le programme a besoin de plusieurs informations pour fonctionner :

- Le nombre de machines *m*
- La durée des *n* tâches (qui peuvent être des entières ou décimales)

Ces données peuvent être renseignées de plusieurs manières :

- Dans le mode « fichier » où l'utilisateur transmet le chemin du fichier qui sera utilisé, dans lequel il y a une unique ligne de la forme « $m{:}n{:}d_1{:}\ d_2{:}\ d_3{:}\dots{:}d_{n-2}{:}d_{n-1}{:}d_n$ » avec *m* le nombre de machines, *n* le nombre de tâches à réaliser et $d_1, d_2\dots$ la durée des tâches.
- Dans le mode « clavier » où l'utilisateur renseigne une ligne de la forme « $m{:}n{:}d_1{:}\ d_2{:}\ d_3{:}\dots{:}d_{n-2}{:}d_{n-1}{:}d_n$ »
  avec *m* le nombre de machines, *n* le nombre de tâches à réaliser et $d_1, d_2\dots$ la durée des tâches.
- Dans le type « I_m » où l'utilisateur ne renseigne que le nombre *m* de machines, puis le programme calcule la durée des 2*m*+1 taches.
- Dans le type « random » où l'utilisateur renseigne :
  - Le nombre de machines
  - Le nombre de tâches à exécuter
  - Les bornes supérieures et inférieures des tâches à réaliser

  Le programme utilisera des durées aléatoires (entre les bornes inférieures et supérieures) pour les *n* tâches à répartir sur les *m* machines.

2.4. Usage

La façon la plus facile d'utiliser le programme est d'accéder au répertoire de l'exécutable puis de le lancer comme tout programme. Cela ouvre un menu principal avec différents choix possibles pour renseigner les données demandées (voir partie précédente « Structure de données »).

```
===========================
--      MIN-MAKESPAN      --
===========================

1. Donner l'instance depuis un fichier
2. Donner l'instance depuis le clavier
3. Générer une instance de type Im
4. Générer des instances aléatoires
5. Quitter
Faites votre choix
>
```

Une seconde façon d'exécuter le programme est de renseigner certaines informations dans les options disponibles en ligne de commande.

Par exemple, pour exécuter les algorithmes avec les données contenues dans un fichier, il suffit de lancer la commande : `java -fichier=chemin_vers_le_fichier minmakespan.minmakespan`

Pour connaitre toutes les options possibles, il suffit de lancer la commande `java -help minmakespan.minmakespan`

```
=====================================
-- Aide a l'utilisation du programme --
=====================================
Objectif : Calculer une duree minimale pour l'execution de N taches sur M machines
Auteurs : VENTE Maxime et HARTLEY Marc
Usage:
  Classique   : java minmakespan.minmakespan <options>
  Random      : java minmakespan.minmakespan -random <-m=M> <-n=N> <-k=K> <-min=MIN> <-max=MAX>
  Interactive : java minmakespan.minmakespan -i
  Fichier     : java minmakespan.minmakespan <-fichier=CHEMIN>
  Im-type     : java minmakespan.minmakespan -Im <-m=M>
  Aide        : java minmakespan.minmakespan -h
Options possible :
  -h -help                : Affiche cette page d'aide
  -m -machines            : nombre de machines a utiliser
  -n -t -taches -tasks    : nombre de taches
  -k -iter -iterations    : nombre d'iterations a executer
  -min                    : duree minimale d'une tache
  -max                    : duree maximale d'une tache
  -random -r              : utilisation du mode aleatoire
  -f -fichier             : chemin vers le fichier utilise pour appliquer les calculs
  -i -interactive         : mode interactif utilise
  -Im                     : utilisation d'une instance type Im
```

2.5. Algorithmes implémentés

Tous les algorithmes sont classés 2-approximation, c'est-à-dire que le résultat est au maximum le double du temps optimal.

2.5.1. LSA

LSA (*List Simulated Annealing*) est l'algorithme intuitif que nous utiliserions : il suffit de prendre chaque tâche dans l'ordre renseigné puis de la faire exécuter par la première machine disponible. Répéter jusqu'à ce qu'il n'y ait plus de tâche inachevée.

2.5.2. LPT

LPT (*Longest Processing Time*) est une variante du LSA dans laquelle la liste des tâches est triée par ordre décroissant de leur durée.

2.5.3. MyAlgo

Cet algorithme mélange une variante de First-Fit avec le LPT.

On suppose qu'on a m machines $\{M_1\,;\ M_2\,;\ \ldots\ M_m\}$. On s'occupe tout d'abord de la machine M_1 dont on affecte des tâches jusqu'à ce que cela lui prenne la moyenne des temps d'exécution par machine ($\frac{\sum_{i=0}^{n} d_i}{m}$), puis on continue l'opération sur M_2, puis M_3, et ainsi de suite jusqu'à M_n. Ensuite on réalise une LPT avec les tâches restantes et en partant du temps d'exécution actuel de chaque machine.

2.6. Résultats du programme

Le programme retourne (sauf cas du mode « random ») les résultats de chaque algorithme avec le ratio calculé avec les bornes du temps optimal. Pour rappel, $T_{opt} \geq \frac{\sum_{i=0}^{n} d_i}{m}$ et $T_{opt} \geq max(d)$

```
============================
--        MIN-MAKESPAN    --
============================

1. Donner l'instance depuis un fichier
2. Donner l'instance depuis le clavier
3. Générer une instance de type Im
4. Générer des instances aléatoires
5. Quitter
Faites votre choix
> 3
Nombre de machines :
> 25
Borne inférieure ``maximum'' = 49.0
Borne inférieure ``moyenne'' = 75.0
Résultat LSA = 111.0
ratio LSA = 1.48
Résultat LPT = 99.0
ratio LPT = 1.32
Résultat myAlgo = 110.0
ratio MyAlgo = 1.4666667
```

Avec le mode « random », on limite l'affichage à des « ratio moyens », c'est-à-dire les ratios calculés sur les *k* itérations demandées.

```
============================
--      MIN-MAKESPAN      --
============================

1. Donner l'instance depuis un fichier
2. Donner l'instance depuis le clavier
3. Générer une instance de type Im
4. Générer des instances aléatoires
5. Quitter
Faites votre choix
> 4
Nombre de machines :
> 25
Nombre de taches :
> 100
Nombre d'itérations :
> 50
Durée minimale d'une tache :
> 1
Durée maximale d'une tache :
> 10
ratio moyen LSA = 1.2245618
ratio moyen LPT = 1.0228169
ratio moyen MyAlgo = 1.0979033
```

3. Discussion sur le programme

3.1. Complexité en temps

3.1.1. Generation d'instances

Random

Pseudo-code :

```
Def doRandom(m : entier, n : entier, k : entier, min : décimal, max : décimal) :
        Initialiser datas tableau 2-D de décimaux
        Pour i allant de 1 à k :
                Initialiser data tableau de décimaux
                data[1] = m
                data[2] = n
                data[3 à 3 + n] = generation de n nombres décimaux aléatoires entre min et
max
                Ajouter data à datas
        Fin Pour
Fin doRandom
```

Complexité :

On réalise une boucle k fois, dans laquelle on génère n nombre décimaux, on a donc une complexité de $O(n * k)$

Im

Pseudo-code :

```
Def doIm(m : entier) :
        Initialiser data tableau de 2m+3 décimaux
        data[1] = m
        data[2] = n
        durée = m
        Pour i allant de 1 à 2n + 1 :
                Si i > 3 et i est pair : ajouter 1 à durée
                Ajouter durée à data
        Fin Pour
Fin doIm
```

Complexité :

On a simplement une boucle de 2n+1 itérations, donc une complexité de $O(n)$

Copie en mémoire des instances

If et Ic

Pseudo-code :

```
Def lectureFichier(fichier : String)
        Récupérer dans ligne la première ligne du fichier fichier
        Découper ligne à chaque « : »
        m = ligne[1]
        n = ligne[2]
        initialiser un tableau de décimaux de taille n + 2, y ajouter m et n
        Pour i allant de 1 à n :
                Tableau[i + 2] = ligne[i + 2]
        Fin Pour
Fin lectureFichier
```

Le code pour l'entrée de l'utilisateur est le même, mais au lieu de lire la première ligne du fichier, lire l'entrée utilisateur.

Complexité :

Une unique boucle allant de 1 à n, donc la complexité est de n.

3.1.2. Algorithmes :

LSA :

Pseudo-code :

```
Def LSA(m : entier, n : entier, taches : tableau de décimaux, machines : tableau de décimaux)
        Pour i allant de 1 à n :
                Indice = indice de min(machines)
                Machines[indice] += taches[i]
        Fin Pour
        Retourner machines
Fin LSA
```

Le paramètre machines permettra de réaliser une LSA après traitement dans MyAlgo.

Complexité :

On a une boucle de 1 à n, dans laquelle on cherche le min de machines. On a une complexité de $O(n * c(min))$ soit $O(n * m)$.

LPT :

Pseudo-code :

```
Def LPT(m : entier, n : entier, taches : tableau de décimaux, machines : tableau de décimaux)
        trier taches dans l'ordre décroissant
        Réaliser LSA(m, n, taches, machines)
Fin LPT
```

Complexité :

On a une complexité de $O(c(tri) + c(LSA))$, soit $O(n^2 + n * m)$. On peut supposer $m \leq n$ car le cas contraire signifie qu'il suffit de retourner le plus grand élément de taches (voir rapport des exercices, question 4). On a alors, pour conclure, une complexité $O(n^2)$

MyAlgo

Pseudo-code :

```
Def MyAlgo(m : entier, n : entier, taches : tableau de décimaux, machines : tableau de décimaux)
        Trier taches dans l'ordre décroissant
        Moyenne = somme(taches)/m
        TachesRestantes = tableau vide de décimaux
        Pour iM allant de 1 à m :
                Pour chaque tache T :
                        Si machines[iM] + T ≤ moyenne :
                                Machines[iM] += T
                        Sinon :
                                Ajouter T à TachesRestantes
                        Fin Si
                        Retirer T de taches
                Fin Pour
        Fin Pour
        Réaliser LPT(m, n, TachesRestantes, machines)
Fin MyAlgo
```

Complexité :

On a une complexité $O(c(premièresAffectations) + c(LPT))$. On sait que $c(premièresAffectations)$ contient 2 boucles imbriquées de m itérations puis n itérations et que $c(LPT) = O(n^2)$, donc MyAlgo a une complexité$O(n * m + n^2)$. Comme vu dans la partie Complexité de LPT, on peut considérer cette complexité égale à $O(n^2)$. En appliquant la LSA plutôt que la LPT en fin d'algorithme, on peut réduire notre complexité à $O(n \times m)$, sans beaucoup changer les résultats.

3.1.3. Comparaison d'algorithmes

Sur Im :

On a vu dans le rapport d'exercices (question 11 et 13) que LSA a un ratio d'approximation de 3/2 et LPT a un ratio de 4/3. MyAlgo a relativement les mêmes résultats que LPT, et tend à un ratio de 4/3 quand m est grand.

Sur Random :

Pour étudier le degré d'approximation sur des listes de taches aléatoires, cela se complexifie car beaucoup de paramètres rentrent en jeu : m, n, min, max. Ici chaque ratio est donné sur 100 itérations.

Pour m = 1000, n = 5000, min = 10, max = 100 :

LSA : 1.23 -- LPT : 1.03 – MyAlgo : 1.06

Pour m = 1000, n = 1500, min = 10, max = 100 :

LSA : 1.47 – LPT : 1.0 – MyAlgo = 1.12

Pour m = 1000, n = 5000, min = 1, max = 2

LSA : 1.15 – LPT : 1.01 – MyAlgo : 1.01

Pour m = 1000, n = 5000, min = 1, max = 100

LSA : 1.25 – LPT : 1.04 – MyAlgo : 1.07

On peut donc voir que rapprocher n de m dégrade le ratio de LSA et MyAlgo tandis que cela améliore celui de LPT.

De plus, augmenter le rapport max/min dégrade tous les ratios, mais MyAlgo en souffre plus que LPT.