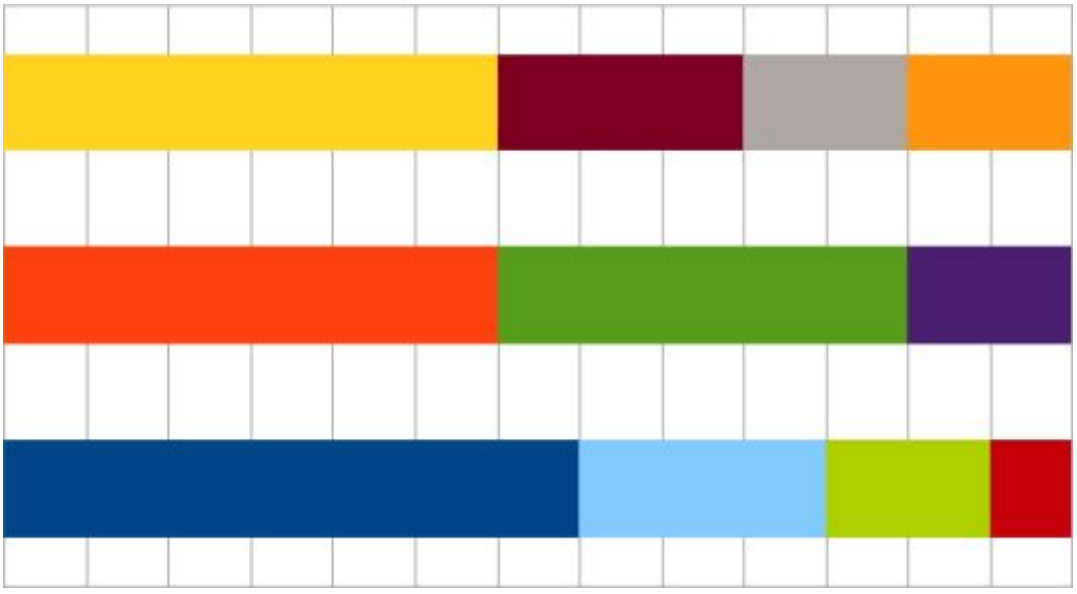
Le problème Min Makespan



Mahier Loïc
Phalavandishvili Demetre

I: Exercice

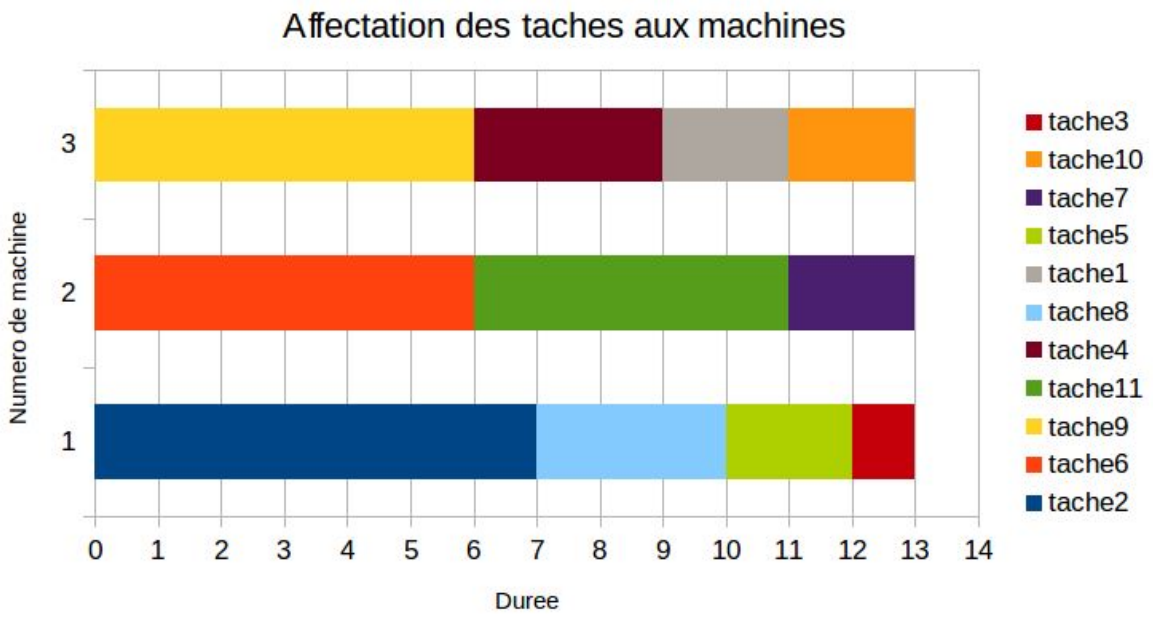
Question 1

Données :

- 3 machines
- 11 tâches
- d1 = 2; d2 = 7; d3 = 1; d4 = 3; d5 = 2; d6 = 6; d7 = 2; d8 = 3; d9 = 6; d10 = 2; d11 = 5.

Par ordre décroissant, on obtient : d2 > d6 > d9 > d11 > d4 > d8 > d1 > d5 > d7 > d10 > d3.

Voici une illustration du résultat obtenu en exécutant l'algorithme LPT :



Question 2

Sur cet exemple, l'optimum est de 13. En effet, toutes les machines s'arrêtent en même temps, pour t = 13. Le ratio d'approximation obtenu par LPT est donc de 1 et est idéal, on ne peut pas faire mieux. (13 tâches, durée max de 13 : 13 / 13 = 1)

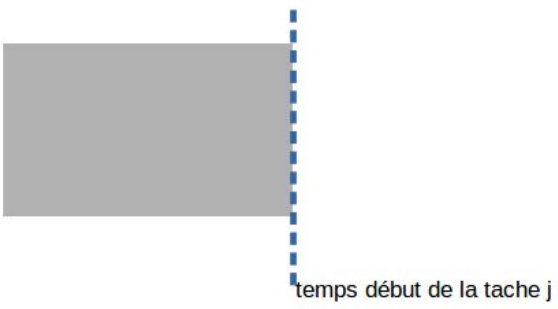
Question 3

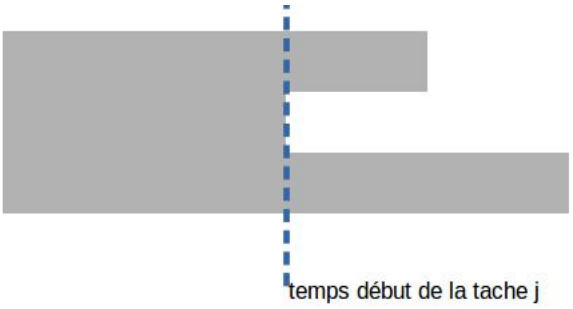
Montrons que : $T_{LPT}(I) \leq \sum_{k \neq j} \frac{d'_k}{m} + d'_j$

On peut dire que : (I) $T_{LPT}(I) = t_{deb} + d'_j$ où :

- t_{deb} : le temps auquel la tâche *j* est démarré sur la machine libre
- d_j : la durée de la tâche *j*

Ainsi, t_{deb} a deux représentations :





En se basant sur la première figure on peut écrire l'égalité suivante :

(II) $t_{deb} = \sum_{k \neq j} \frac{d'_k}{m}$

En se basant sur la deuxième figure on peut écrire l'inégalité suivante :

(III) $t_{deb} < \sum_{k \neq j} \frac{d'_k}{m}$ car la moyenne entre les différents temps de fin des machines va se situer à droite par rapport à t_{deb}.

L'expression (I) implique que $t_{deb} = T_{LPT}(I) - d_j$, puis à partir de ce résultat on obtient les deux expressions suivantes :

- (IV) $T_{LPT}(I) - d'_j = \sum_{k \neq j} \frac{d'_k}{m}$
- (V) $T_{LPT}(I) - d'_j < \sum_{k \neq j} \frac{d'_k}{m}$

De (IV) on peut déduire que : (VI) $T_{LPT}(I) = \sum_{k \neq j} \frac{d'_k}{m} + d'_j$

De (V) on peut déduire que : (VII) $T_{LPT}(I) < \sum_{k \neq j} \frac{d'_k}{m} + d'_j$

Les expressions (VI) et (VII) implique :

(VIII) $T_{LPT}(I) \leq \sum_{k \neq j} \frac{d'_k}{m} + d'_j$

Question 4

Supposons que $n \leq m$, où n est le nombre de tâche et m le nombre de machine. Cela signifie que chaque machine aura au plus une tâche à effectuer.

Donc d'un part on peut dire que :

$$T_{LPT}(I) = max_{1 \leq i \leq n}\, d'_i$$

De l'autre part on peut dire que :

$$T_{OPT}(I) = max_{1 \leq i \leq n}\, d'_i$$

On en déduit que LPT est toujours optimal pour les instants $n \leq m$

Question 5

Supposons à présent que $n >= m + 1$. Cela signifie qu'on a au moins $m + 1$ tâches, soit forcément plus de tâche que de machine.

Montrons que : $T_{OPT}(I) >= 2 * d'_{m+1}$

Dans notre cas c'est vrai, en effet : $13 >= 2 * 3$.

Ainsi, il s'agit de la $m + 1^{ème}$ tâche, de fait elle sera donc exécutée en 2ème sur une machine. En effet, dans notre cas par exemple : ce sera la 4ème tâche pour 3 machines. De plus, la tâche qui sera exécutée sur cette machine avant (avant la $m + 1^{ème}$ tâche), sera au moins aussi grande, car classée avant (l'ordre est décroissant). Donc le temps optimal dans

le cas ou cette tâche (la $m + 1^{ème}$) est la dernière sera au minimum égal à la somme de ce temps plus celui de la tâche exécutée auparavant sur cette machine.

Donc $T_{OPT}(I) >= 2 * d'_{m+1}$.

Question 6

Montrer que l'on distingue deux cas :

- $T_{LPT}(I) = T_{OPT}(I)$

Pour ce cas, on distingue les deux sous-cas suivant :

- ❏ Le premier cas correspond à la situation où le nombre de tâches est égal au nombre de machines. Cela signifie qu'on a une tâche par machine à exécuter. Donc d'un côté on peut en déduire que $T_{LPT}(I) = d'_1$, où d'_1 est la tâche la plus longue. D'un autre côté on peut dire que $T_{OPT}(I) = d'_1$, car d'_1 est la tâche la plus longue. Au final on en déduit que $T_{LPT}(I) = T_{OPT}(I)$

- ❏ La deuxième cas correspond à la situation où la répartition des tâches par machine est idéale. Et pour chaque machine la durée totale des tâches exécutées est la même. (exemple : voir la question 1)

- $j \geq m + 1$

- ❏ C'est la configuration dans laquelle nous avons plus de tâches que de machines. Cette configuration correspond à la configuration décrite dans la question précédente.

Question 7

D'après le question 5, on sait que :

$$T_{OPT}(I) \geq 2 \times d'_{m+1}$$
$$\Leftrightarrow \frac{1}{2} T_{OPT}(I) \geq d'_{m+1}$$

D'après le question 6, on sait que *j* est la tâche qui se termine en dernier, on peut alors en déduire que :

$$d'_{m1+} \geq d'_j$$

On peut donc en déduire que :

$$\tfrac{1}{2} T_{OPT}(I) \geq d'_j$$

D'autre part, d'après la question 3 nous savons que :

$$T_{LPT}(I) \leq \sum_{k \neq j} \frac{d'_k}{m} + d'_j \quad \text{, ce cas correspond au cas } j \geq m+1$$

$$\Leftrightarrow \quad T_{LPT}(I) - d'_j \leq \sum_{k \neq j} \frac{d'_k}{m} + d'_j - d'_j$$

$$\Leftrightarrow \quad T_{LPT}(I) - d'_j \leq \sum_{k \neq j} \frac{d'_k}{m}$$

D'après l'expression (I) de la question 3 on peut écrire :

$$\Leftrightarrow \quad t_{deb} \leq \sum_{k \neq j} \frac{d'_k}{m}$$

En effectuant tous les manipulations précédente, nous avons ramené notre inéquation au cas où on a autant de machine que de taches, donc en utilisant la question 4, on en déduit que :

$$t_{deb} \leq \sum_{k \neq j} \frac{d'_k}{m} \leq T_{OPT}(I)$$

Au final, on obtient :

(I) $\quad d'_j \leq \tfrac{1}{2} T_{OPT}(I)$

(II) $\quad \sum_{k \neq j} \frac{d'_k}{m} \leq T_{OPT}(I)$

(I) + (II) $\quad \sum_{k \neq j} \frac{d'_k}{m} + d'_j \leq \tfrac{3}{2} T_{OPT}(I)$

D'après le (VIII) de la question 3, on en déduit que :

$$T_{LPT}(I) \leq \frac{3}{2} T_{OPT}(I)$$

Question 8

On en déduit que $r = \frac{3}{2}$.

II : Programmation

Introduction

Passons à présent à la partie programmation du distanciel. Nous avons choisi de la réaliser en C++ pour plusieurs raisons. Premièrement parce que c'est un language très rapide, ce qui nous paraissait important étant donné que le but était d'implémenter des algorithmes d'approximation sur des instance plus ou moins grandes et plus ou moins nombreuses. Deuxièmement parce que le C++ avait l'avantage d'être un langage assez familier pour nous puisque nous l'utilisons couramment à la faculté.

L'objectif de la partie programmation du distanciel était d'implémenter plusieurs algorithmes, de les tester et de fournir les résultats à l'utilisateur. Aussi nous ne nous attarderons pas sur certaines parties du code tels que la réalisation du menu, la manière de lire et écrire dans les fichiers ou encore la création des instances. Nous argumenterons uniquement sur nos choix pour modéliser une instance et implémenter les algorithmes. Ce après quoi nous discuterons des résultats obtenus.

Modéliser une instance

Nous avions plusieurs possibilités pour modéliser une instance. Parmis celles-ci deux nous on fait hésiter, l'utilisation des classes (propre au C++) et celle d'une structure propre au C). Nous avons choisi d'opter pour une structure par praticité, en effet nous souhaitons uniquement modéliser une instance, nous n'avons pas besoins qu'elle ai ses fonctions propres ou que ses attributs soient privés. De fait dans notre contexte il nous semblait plus simple d'utiliser une structure.

Ainsi, nous avons une structure "Instance" qui contient deux entiers, "nbMachine" et "nbTache", et un tableau de taille "nbTache" contenant les durées de chacune des tâches. La durée étant donc associée à la tâche "indicée" par le tableau : T[4] = 12 signifie que la tâche 4 a pour durée 12.

Implémentation des algorithmes

Les trois algorithmes ont été implémentés de la même manière, de sorte qu'ils aient la même signature. En effet nous avons trois fonctions avec le même paramètre en entrée et le même élément en sortie, à savoir un tableau "d'instance" en entrée et un tableau d'entier en sortie. Cela simplifie les choses pour les méthodes qui vont retourner les résultats à l'utilisateur.

Les algorithmes LSA et LPT sont identiques à une chose près : dans le cas de LPT nous trions le tableau de durée par ordre décroissant. Ces deux fonctions procèdent donc comme suit : elles parcourent toutes les instances passées en paramètre. Pour chacune des instances, elles trient les durée ou non (comme expliqué ci-dessus). Il faut également créer un tableau qui aura pour taille notre nombre de machine : les indices représentent la machines, et les valeurs aux indices représentent la durée totale des tâches qui lui sont affectées. Ainsi, nous parcourons notre tableau de durée et nous affectons à la première machine disponible la tâche courante (la machine qui à donc la valeur la plus petite et le plus petit indice). Ce qui est modélisé par l'addition de la durée de cette tâche à la valeur de la machine. Une fois toutes les tâches affectées, il nous suffit de regarder quelle machine à la valeur la plus grande. Cela correspond au temps auquel finit la dernière tâche. Temps que nous ajoutons à un tableau qui va contenir le temps maximal de toutes nos instances. C'est ce tableau que nous retournons une fois toutes les instances parcourues.

Notre algorithme est basé sur la répartition des tâches par machine en fonction de la moyenne de la durée totale par le nombre de machine. Une fois la moyenne calculée, on distribue les tâches sur la première machine en essayant de se rapprocher de la moyenne ou de la dépasser au minimum. On applique la même méthode sur les autres machines. La complexité de notre algorithme implémentée est en $\Theta(n)$, car même si on a deux boucles *while* imbriqués on parcourt qu'une fois chaque tâche. Ceci est garantie par l'affectation suivante *i = j* après la deuxième boucle *while*, cela signifie qu'on continue la première boucle à partir de l'indice *j+1*.

Analyse des résultats

En analysant les résultats, on peut dire que pour certaines instances notre algorithme fait mieux que LSA et quelque fois même mieux que LPT, mais c'est rare. Prenons l'exemple de la configuration suivante :

Nombre de machines	5	Durée minimum	1
Nombre des tâches	20	Durée maximum	10
Nombre d'instance		50	

Nous obtenons les ratios d'approximation suivants :

Ratio d'approximation moyen LSA	2,12
Ratio d'approximation moyen LPT	1,9
Ratio d'approximation moyen myAlgo	2,32

On constate donc que dans la majorité des cas, notre algorithmes est moins performant que LSA et LPT. Dans les faits, il est très difficile, voir impossible de faire mieux que LPT qui est très souvent optimal ou du moins s'en approche. Cependant on pourrait faire mieux que LSA.

Conclusion

Finalement, ce qu'on peut retenir de ce distanciel est qu'il existe déjà de très bon algorithme pour résoudre Min-Makespan, tel que LPT. Cet algorithme est souvent optimal ou s'en approche. Il existe également d'autres algorithmes comme LSA, moins performant mais tout de même intéressant.

On conclura, pour finir, sur le fait qu'il n'est pas évident d'implémenter un algorithme mieux que LPT, que ce soit en terme de résultat obtenu ou de complexité de l'implémentation (en temps et en espace) .