

2 P-RAM : Rupture de symétrie déterministe

On veut concevoir un algorithme EREW qui permet de sélectionner « beaucoup » d'objets dans une liste chaînée sans jamais choisir deux objets adjacents dans la liste. Chaque objet est associé à un processeur mais le numéro du processeur n'est pas relié à l'ordre des éléments dans la liste.

▷ **Question 4** *Concevoir un algorithme qui permet de sélectionner un nombre optimal d'objets en temps $O(\log n)$ sur une EREW.*

Dans la suite on va montrer qu'il est possible d'extraire « beaucoup » d'éléments en un temps $O(\log^* n)$, où

$$\log^* n = \min\{i \mid \log^i n \leq 1\}$$

Dans l'expression précédente, \log^i représente la composition de i fois la fonction \log . La fonction \log^* a une croissance très lente, puisque $\log^*(2^{65536}) = 5$.

Nous allons maintenant préciser le sens de « beaucoup » à partir de la notion d'ensemble indépendant maximal.

Définition 1. Un ensemble de sommets V' d'un graphe $G = (V, E)$ est indépendant ssi

$$\forall (a, b) \in E, \text{ au plus un élément de } \{a, b\} \text{ est dans } V'$$

Un ensemble de sommets V' d'un graphe $G = (V, E)$ est indépendant et maximal ssi l'ajout de tout sommet à V' en fait un ensemble non indépendant.

Dans ce qui suit, notre objectif est d'arriver à extraire un ensemble indépendant maximal de la liste en temps $O(\log^* n)$. Pour cela, on va commencer par colorier la liste (avec 6 couleurs). On va construire un algorithme qui part d'une n -coloration (où la couleur de chaque objet est déterminée par la couleur du processeur qui lui est associé) et qui diminue à chaque étape le nombre de couleurs utilisées.

▷ **Question 5** *Donner un algorithme astucieux pour diminuer le nombre de couleurs utilisées tout en gardant une coloration (on pourra raisonner sur le codage binaire des couleurs).*

▷ **Question 6** *Pourquoi obtient-on 6 couleurs ? Montrer qu'on obtient 6 couleurs après $O(\log^* n)$ étapes.*

3 Ordonnancement

3.1 Ordonnancement sans communication

On veut ordonnancer un ensemble de n tâches indépendantes T_1, T_2, \dots, T_n . La durée de T_i est p_i . On suppose que l'ordonnancement débute au temps $t = 0$, et on note C_i la date de la fin de l'exécution de la tâche T_i .

▷ **Question 7** *On dispose d'un seul processeur dans cette question.*

1. *Quel est l'ordre d'exécution optimal si l'objectif est de minimiser la somme des dates de fin $\sum_{i=1}^n C_i$?*
2. *Quel est l'ordre d'exécution optimal si l'objectif est de minimiser la somme pondérée des dates de fin, i.e. $\sum_{i=1}^n w_i.C_i$, où w_i est un poids positif associé à la tâche T_i ?*

▷ **Question 8** *On dispose de $p \geq 2$ processeurs identiques dans cette question. Quel est l'ordre d'exécution optimal si l'objectif est de minimiser la somme des dates de fin $\sum_{i=1}^n C_i$?*

3.2 Comparaison d'heuristiques

On considère le modèle avec coût de communication du cours. Pour le graphe de tâches de la Figure 4, les poids des tâches sont indiqués entre parenthèses, et ceux des arêtes le long de celles-ci.

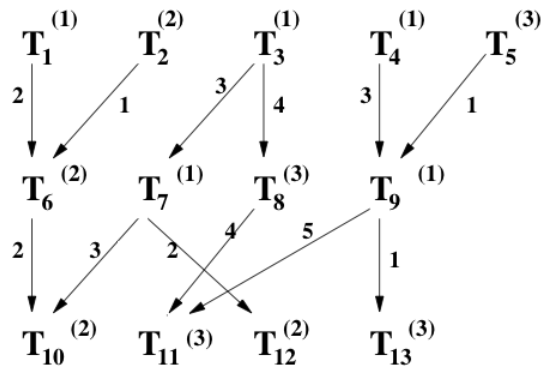


FIG. 4 – Graphe de tâches à ordonnancer.

▷ **Question 9** *On suppose disposer d'un nombre illimité de processeurs. Ordonnancer le graphe avec l'heuristique du chemin critique modifié, l'heuristique de Kim et Browne, puis l'heuristique de Sarkar.*

▷ **Question 10** *On suppose maintenant disposer de seulement $p = 3$ processeurs. Quel est l'ordonnancement optimal ?*

4 Réponses aux exercices

▷ Question 1, page 1

Commençons par remarquer que pour toute matrice bande M , on a :

$$M_{ij} \neq 0 \Rightarrow 1 - q_M \leq j - i \leq p_M - 1.$$

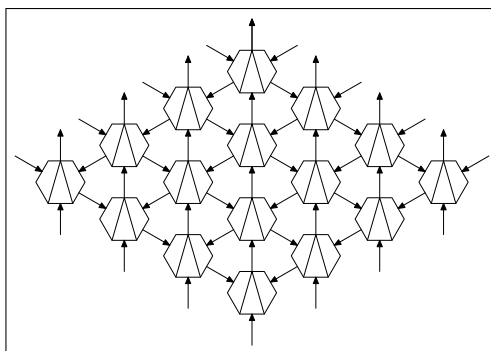
Étant donné que $c_{ij} = \sum_k a_{ik}b_{kj}$, observons les éléments de cette somme :

$$\begin{aligned} a_{ik} \neq 0 &\Rightarrow 1 - q_A \leq k - i \leq p_A - 1 \\ b_{kj} \neq 0 &\Rightarrow 1 - q_B \leq j - k \leq p_B - 1 \\ a_{ik}b_{kj} \neq 0 &\Rightarrow 2 - (q_A + q_B) \leq j - i \leq (p_A + p_B) - 2 \end{aligned}$$

C est bien une matrice bande (p_C, q_C) , avec $p_C = p_A + p_B - 1$ et $q_C = q_A + q_B - 1$.

▷ Question 2, page 1

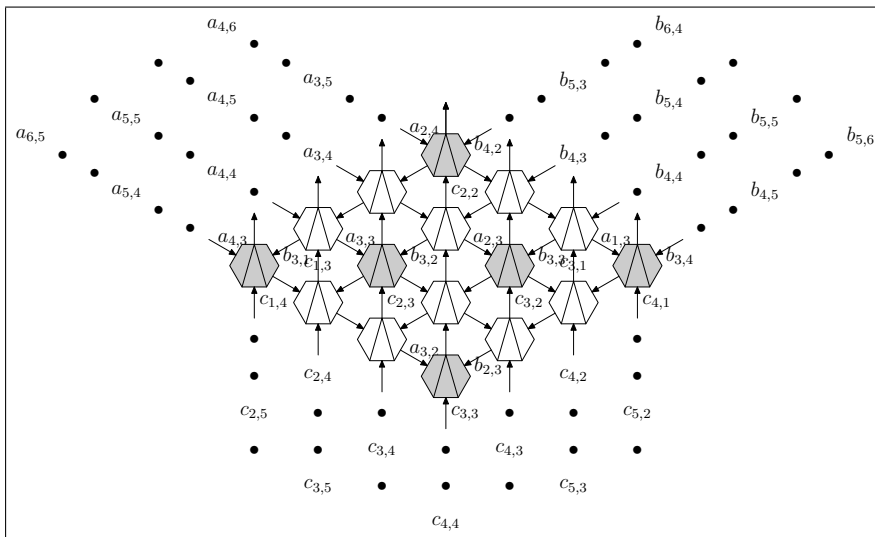
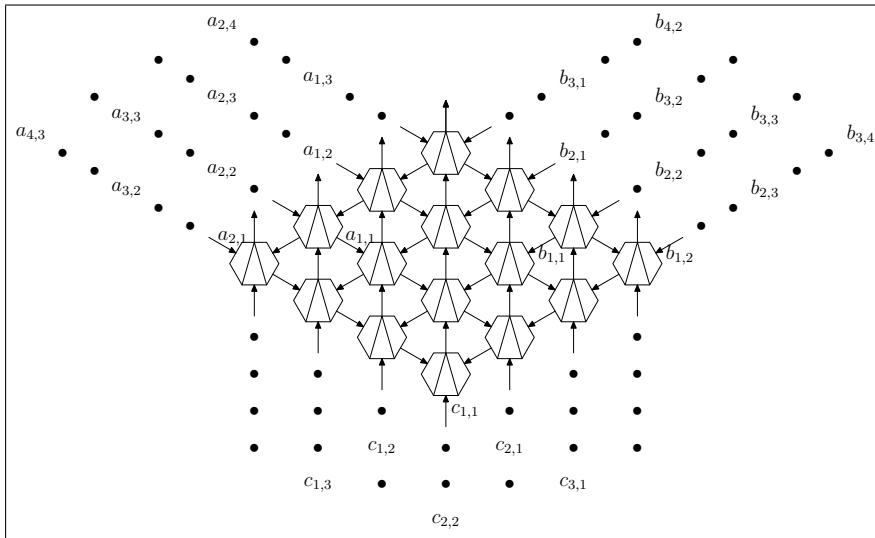
Nous allons traiter le cas du produit d'une matrice bande $(3, 2)$ par une matrice bande $(2, 3)$. La matrice résultante étant une matrice bande $(4, 4)$, il est assez naturel d'utiliser le réseau suivant :



Quelles sont les contraintes auxquelles on fait face ?

1. Tout d'abord, étant donné la structure des matrices, il est impossible de les faire rentrer par ligne si on s'impose un réseau dont la taille est indépendante de la taille des matrices. Il est bien plus naturel de les faire rentrer par diagonales. Au vu du type de cellule donné, il est raisonnable de faire entrer A par la gauche, B par la droite et C par le bas. On pourra vérifier que les dimensions du réseau sont bien compatibles avec la structure de A , B et C .
2. La diagonale de C étant celle qui requiert le plus de calcul, elle doit circuler vers le haut en passant au milieu du réseau. Considérons par exemple le calcul de c_{33} . On a $c_{33} = a_{32}b_{23} + a_{33}b_{33} + a_{34}b_{43} + a_{35}b_{53}$. Supposons que c_{33} soit en (entrée de la cellule) $(0, 0)$ au temps t . Alors a_{32} et b_{23} sont aussi en $(0, 0)$ au même top t . Au temps $t + 1$, c_{33} est en $(1, 1)$, donc a_{33} et b_{33} aussi. a_{33} était donc en $(2, 1)$ au temps t et b_{33} en $(1, 2)$ au temps t . Au temps $t + 2$, c_{33} est en $(2, 2)$ et en refaisant le même raisonnement, on en déduit que a_{34} doit être en $(4, 2)$ au temps t et b_{43} en $(2, 4)$. On sait donc que si le premier élément d'une ligne de A est en $(0, 0)$ au temps t , le d -ième élément de cette ligne est en $(2d, d)$ au temps t ; si le premier élément d'une colonne de B est en $(0, 0)$ au temps t , le d -ième élément de cette colonne est en $(d, 2d)$ au temps t .
3. Il faut maintenant savoir à quelle vitesse les éléments d'une même diagonale rentrent dans le réseau. Regardons par exemple l'élément de A qui suit a_{32} , c'est-à-dire a_{43} . Cet élément a_{43} doit rencontrer b_{33} pour participer au calcul de $c_{43} = a_{43}b_{33} + a_{44}b_{43} + a_{45}b_{53}$. b_{33} étant en $(1, 0)$ au temps $t + 3$, c_{43} et a_{43} doivent également y être. On en déduit donc que a_{43} est en $(4, 0)$ au temps t et c_{43} en $(-1, -2)$. Les éléments d'une même diagonale vont donc rentrer tous les trois tops dans le réseau.

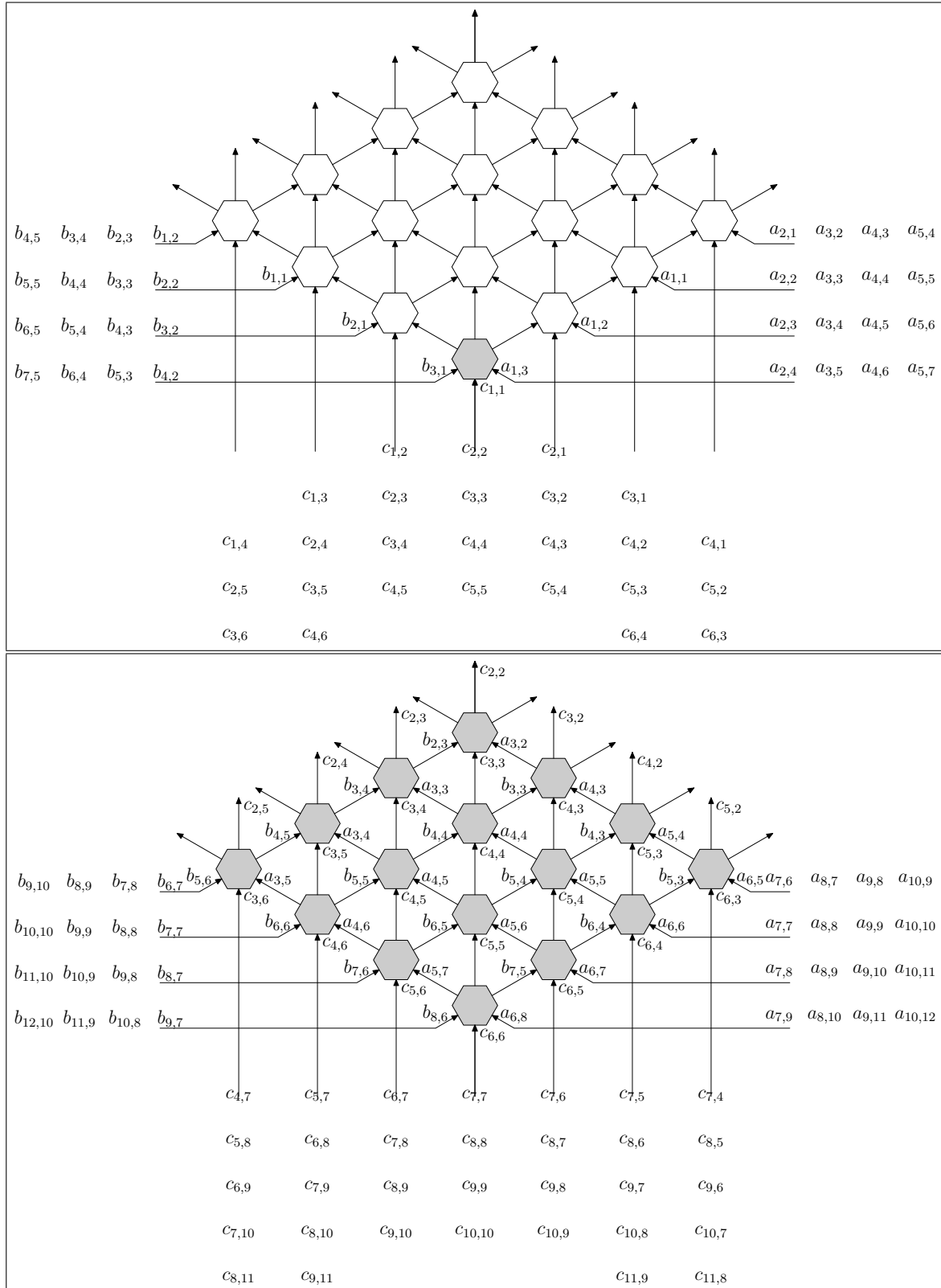
On peut déduire des remarques précédentes le réseau suivant : À titre d'exemple, l'état du réseau au temps $t = 7$, est représenté ci-dessous.



Chaque cellule n'est active qu'une fois sur trois et le temps nécessaire pour effectuer le produit de matrice est donc de $3n + O(p + q)$. On remarquera qu'il est possible de pipeliner 3 produits de matrices en entrelaçant les matrices pour augmenter le rendement du réseau, dû à Kung et Leiserson [?].

▷ Question 3, page 1

En effectuant le même type d'analyse que précédemment, on peut construire le réseau suivant :



La figure montre que le rendement est bien meilleur que le précédent : les coefficients rentrent tous les tops dans le réseau, et le produit est effectué en temps $n + O(p+q)$. Ce réseau est dû à Weiser et Davis [?].

▷ **Question 4, page 2**

Il suffit d'utiliser l'algorithme vu en cours pour la distance à la fin de la liste. Chaque processeur détermine en temps $O(\log n)$ sa distance à la fin de la liste. Si cette distance est paire, son objet est sélectionné, sinon il ne l'est pas.

4 Réponses aux exercices

▷ Question 12, page 2

Pour obtenir une coloration en un temps $O(\log^* n)$, il faut que le nombre de couleurs nécessaire r_{k+1} soit de l'ordre de $\log(r_k)$. Considérons deux nœuds successifs a et b à l'étape k dont les couleurs sont respectivement codées par

$$C^k(a) = \langle a_{r_k-1}, a_{r_k-2}, \dots, a_0 \rangle \text{ et } C^k(b) = \langle b_{r_k-1}, b_{r_k-2}, \dots, b_0 \rangle.$$

À l'étape $k+1$, on définit le codage de a par :

$$C^{k+1}(a) = \langle \langle i \rangle, a_i \rangle$$

où i est le plus petit indice tel que $a_i \neq b_i$ et $\langle i \rangle$ le codage binaire de i . Comme $i \in [0, r_k - 1]$, on vérifie que la longueur du codage binaire de i est $\leq \lceil \log r_k \rceil$. Ainsi si on note

$$C^{k+1}(a) = \langle a_{r_{k+1}-1}, a_{r_{k+1}-2}, \dots, a_0 \rangle$$

le codage de a après la $k+1^e$ étape, on a

$$r_{k+1} = \lceil \log r_k \rceil + 1$$

Comme la couleur d'un nœud est définie à partir de la couleur de son successeur, il est nécessaire de définir la couleur du dernier nœud d de façon particulière, et on pose

$$C^{k+1}(d) = \langle 0^{r_{k+1}}, d_0 \rangle.$$

Pour vérifier la correction de l'algorithme, il suffit de vérifier qu'on obtient bien un coloriage à chaque étape, c'est-à-dire que si deux nœuds successifs a et b ont une couleur différente à l'étape k , ils ont bien une couleur différente à l'étape $k+1$.

Si $C^k(a)$ et $C^k(b)$ sont différents, le plus petit indice i où les deux codages diffèrent est bien défini. Soit $C^{k+1}(a) = \langle \langle i \rangle, a_i \rangle$ et $C^{k+1}(b) = \langle \langle j \rangle, b_j \rangle$ les codages des couleurs de a et b après l'étape $k+1$. Si $i \neq j$, alors les codages sont différents et si $i = j$ alors $a_i \neq b_i$ par construction.

On a donc construit un algorithme qui permet à chaque étape d'obtenir un nouveau coloriage des nœuds de la liste.

▷ Question 13, page 3

Si r_k vaut 3, on vérifie que r_{k+1} vaut également 3, et le nombre de couleurs utilisées stagne. Plus précisément, comme la position à laquelle les codages des couleurs de deux nœuds consécutifs peut être 0, 1 ou 2, les seuls codages qu'on peut obtenir sont $\langle 000 \rangle$, $\langle 001 \rangle$, $\langle 010 \rangle$, $\langle 011 \rangle$, $\langle 100 \rangle$, $\langle 101 \rangle$, ce qui conduit bien à 6 couleurs différentes. Pour obtenir le 6 coloriage promis, on va utiliser l'algorithme suivant :

- **Tant que** $r \geq 5$, Effectuer une étape de l'algorithme de coloriage.
- Effectuer 3 étapes de l'algorithme de coloriage

Les 3 étapes finales nous assurent qu'on obtient bien un 6-coloriage. Pour étudier la complexité de l'algorithme, il est nécessaire de montrer le lemme suivant :

Dans la boucle **Tant que** on a $r_k \geq 5$ et $r_k \leq \lceil \log n \rceil + 2$. Cette propriété est vérifiée de façon immédiate pour $k=1$, puisque $r_1 \leq \lceil \log n \rceil$. Supposons $r_k \geq 5$ et $r_k \leq \lceil \log^k n \rceil + 2$. On vérifie que $r_{k+1} =$

$$\lceil \log r_k \rceil + 1 \leq \lceil \log(\lceil \log^k n \rceil + 2) \rceil + 1 \leq \lceil \log(\log^k n + 3) \rceil + 1 \leq \lceil \log(2 \log^k n) \rceil + 1 \leq \lceil \log(\log^k n) + 1 \rceil + 1 \leq \lceil \log^{k+1} n \rceil + 2$$

Soit $m = \log^* n$. On vérifie que $\log^{m-1} n + 2 \leq 4$ et donc le nombre d'itérations de la boucle **Tant que** est inférieure à $m-2$. Le nombre d'étapes total de recoloriage de l'algorithme est $m+1$.

▷ **Question 5, page 2**▷ **Question 6, page 2**▷ **Question 7, page 3**

1. On trie les tâches par p_i croissant, et on ordonnance dans cet ordre.

Démonstration. Supposons qu'il existe un ordonnancement optimal qui ne vérifie pas notre ordre, c'est-à-dire qui ordonnance consécutivement deux tâches T_i, T_j avec $p_j < p_i$ et que T_i commence au temps t . La contribution de ces tâches à l'objectif est $2t + 2p_i + p_j$. Si on ordonnance plutôt ces deux tâches dans l'ordre T_j, T_i (sans rien changer d'autre à l'ordonnancement), la contribution de ce couple de tâches sera $2t + 2p_j + p_i < 2t + 2p_i + p_j$, donc l'ordonnancement modifié est strictement meilleur que l'optimal, ce qui n'est pas possible. Donc l'optimal correspond à ordonner les tâches dans l'ordre des p_i croissant. ■

2. Idem, mais en triant par p_i/w_i croissant.

De la même façon, la contribution des tâches T_i, T_j ordonnées dans cet ordre au temps t est :

$$S_i = (w_i + w_j)(t + p_i) + w_j p_j$$

En inversant T_i et T_j on a une contribution :

$$S_j = (w_i + w_j)(t + p_j) + w_j p_i$$

On a

$$\frac{S_i - S_j}{w_i w_j} = \frac{p_i}{w_i} - \frac{p_j}{w_j}$$

Donc la plus faible contribution est obtenue en commençant par ordonner la tâche avec p/w minimum.

▷ **Question 8, page 3**

Voici le schéma de la réponse :

- Ajouter des tâches de durée nulle tel que le nombre de tâches total n soit un multiple de p .
- Allouer les tâches cycliquement au processeur, dans l'ordre des p_i croissants. Montrer que cet algorithme est optimal parmi les ordonnancements équilibrés (c'est-à-dire les ordonnancements tels que tous les processeurs exécutent le même nombre de tâches).
- Montrer que les ordonnancements équilibrés sont dominants, donc que l'algorithme précédent est optimal.