

# Ordonnancement

## 1 Ordonnancement sans coûts de communications

### 1.1 Prérequis de complexité

**Définition 1** ( $\rho$ -approximation). Soit  $\mathcal{P}$  un problème d'optimisation combinatoire dont la fonction objectif  $f_{\mathcal{P}}$  est à valeurs entières. Si on note  $OPT(I)$  une solution optimale du problème  $\mathcal{P}$  pour l'instance  $I$ , on dira qu'un algorithme polynomial  $A$  est une  $\rho$ -approximation de  $\mathcal{P}$  si et seulement si  $\forall I : f_{\mathcal{P}}(A(I)) \leq \rho f_{\mathcal{P}}(OPT(I))$ .

**Théorème 1** (Théorème d'impossibilité). Soit  $\mathcal{P}$  un problème d'optimisation combinatoire dont la fonction objectif  $f_{\mathcal{P}}$  est à valeurs entières et soit  $c$  un entier positif. Si le problème de décision associé à  $\mathcal{P}$  et à la valeur  $c$  est NP-complet, alors pour tout  $\rho < (c + 1)/c$  il n'existe pas de  $\rho$ -approximation de  $\mathcal{P}$  (à moins que  $P=NP$ ).

▷ **Question 1** *Démontrer le théorème d'impossibilité.*

Nous rappelons trois problèmes NP-complets classiques qui pourront être utilisés pour démontrer la difficulté de nos problèmes d'ordonnancement :

**Définition 2** (2-Partition). Étant donné un ensemble  $\mathcal{I}$  de  $n$  nombres  $a_1, \dots, a_n$ , trouver une partition de  $\mathcal{I}$  en deux ensembles  $\mathcal{I}_1$  et  $\mathcal{I}_2$  tels que  $\sum_{i \in \mathcal{I}_1} a_i = \sum_{i \in \mathcal{I}_2} a_i$ .

**Définition 3** (Cliques). Étant donné un graphe  $G = (V, E)$  et un entier  $k$ , trouver un sous-ensemble  $C$  de  $V$  de taille  $k$  tel que pour tout  $u, v \in C$ ,  $(u, v) \in E$ .

**Définition 4** (3-Dimensional-Matching (3DM)). Étant donné trois ensembles  $A = \{a_1, \dots, a_n\}$ ,  $B = \{b_1, \dots, b_n\}$  et  $C = \{c_1, \dots, c_n\}$  ainsi qu'un ensemble  $F = \{T_1, \dots, T_p\}$  de triplets de  $A \times B \times C$ , trouver un sous-ensemble  $F'$  de  $F$  tel que tout élément de  $A \cup B \cup C$  apparaît dans exactement un triplet de  $F'$ .

### 1.2 Tâches indépendantes de durées différentes

Si les tâches sont identiques et indépendantes, le problème est clairement polynomial. En revanche, si les tâches sont de durées différentes et indépendantes, le problème est NP-complet (au sens faible). Mais il existe une  $4/3$ -approximation pour ce problème, ce qui améliore le résultat général pour les algorithmes de liste qui sont toujours des 2-approximations.

Soient  $p$  machines identiques et  $n$  tâches  $(T_i)_{1 \leq i \leq n}$  indépendantes. On cherche donc à définir un ordonnancement  $\sigma$  qui attribue à chaque tâche  $T_i$  une machine  $\mu(T_i)$  et une date de début d'exécution  $\tau(T_i)$  sachant que la durée de la tâche  $T_i$  est  $w(T_i)$ . On cherche à minimiser  $D(\sigma) = \max_{1 \leq i \leq n} (\tau(T_i) + w(T_i))$ .

▷ **Question 2** *En supposant que  $D_{opt} < 3w(T_i)$  pour tout  $i$ , montrer que  $n \leq 2p$  et donner un algorithme polynomial permettant de calculer un ordonnancement de durée minimale.*

▷ **Question 3** *On s'intéresse à l'ordonnancement de liste suivant : dès qu'une machine est libre, on lui affecte la tâche de durée maximale parmi les tâches non encore ordonnancées. Vérifier l'inégalité :*

$$D(\sigma) \leq D_{opt} + \left( \frac{p-1}{p} \right) d,$$

où  $d$  est la durée d'une tâche se terminant à l'instant  $D(\sigma)$ . En déduire l'inégalité :

$$D_{opt} \leq D(\sigma) \leq \left( \frac{4}{3} - \frac{1}{3p} \right) D_{opt}.$$

### 1.3 Tâches identiques avec contraintes de précédence

On cherche à ordonnancer avec  $p$  processeurs identiques un ensemble de  $n$  tâches  $(T_i)_{1 \leq i \leq n}$  unitaires (de durée 1) et reliées par des contraintes de dépendances  $\prec$ .

▷ **Question 4** *Montrer que décider de l'existence d'un ordonnancement dont le temps d'exécution est 3 est un problème NP-complet. (Indication : on pourra se ramener au problème de la clique.)*

▷ **Question 5** *En utilisant le théorème d'impossibilité, donner des résultats d'existence ou d'inexistence d'algorithmes d'approximation pour ce problème d'ordonnancement.*

## 2 Ordonnancement d'un graphe FORK (avec communications)

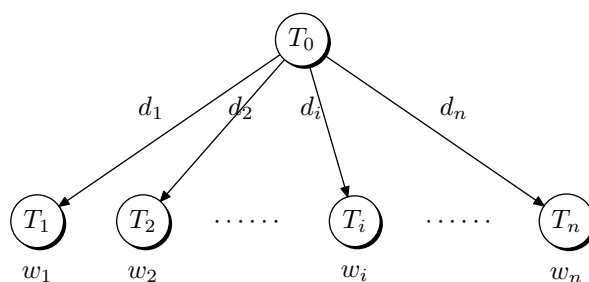


FIG. 1 – Graphe de FORK à  $n$  fils.

**Définition 5** (FORK à  $n$  fils). Un graphe FORK à  $n$  fils est un graphe de tâches à  $n+1$  nœuds étiquetés par  $T_0, T_1, \dots, T_n$ , comme illustré figure 1. Il y a une arête entre le nœud  $T_0$  et chacun de ses fils  $T_i$ ,  $1 \leq i \leq n$ . Chaque nœud possède un poids  $w_i$  qui représente le temps de traitement de la tâche  $T_i$ . Chaque arête  $(T_0, T_i)$  possède aussi un poids correspondant au volume de données échangées  $d_i$  si la tâche  $T_0$  et la tâche  $T_i$  ne sont pas traitées sur le même processeur.

On suppose d'abord disposer d'une infinité de processeurs identiques et multi-port (qui peuvent initier plusieurs envois simultanés). On définit le problème d'optimisation suivant :

**Définition 6** (FORK-SCHED- $\infty(G)$ ). Étant donné un graphe FORK  $G$  à  $n$  fils et un ensemble infini de processeurs identiques, quelle est la durée de l'ordonnancement  $\sigma$  qui minimise le temps d'exécution ?

▷ **Question 6** *Donner un algorithme polynomial résolvant FORK-SCHED- $\infty$ .*

On s'intéresse maintenant au même problème avec un nombre borné de processeurs :

**Définition 7** (FORK-SCHED-BOUNDED( $G, p$ )). Étant donné un graphe FORK  $G$  à  $n$  fils et un ensemble de  $p$  processeurs identiques, quelle est la durée de l'ordonnancement  $\sigma$  qui minimise le temps d'exécution ?

▷ **Question 7** *Montrer que le problème de décision associé à FORK-SCHED-BOUNDED est NP-complet.*

On revient enfin au problème avec une infinité de processeurs identiques, mais on suppose désormais qu'un processeur ne peut communiquer qu'avec un seul processeur à la fois (modèle 1-port).

**Définition 8** (FORK-SCHED-1-PORT- $\infty(G)$ ). Étant donné un graphe FORK  $G$  à  $n$  fils et un ensemble infini de processeurs 1-port identiques, quelle est la durée de l'ordonnancement  $\sigma$  qui minimise le temps d'exécution ?

▷ **Question 8** *Démontrer que le problème de décision associé à FORK-SCHED-1-PORT- $\infty$  est NP-complet. (Indication : on pourra se ramener à 2-Partition-Eq, une variante de 2-Partition où les deux partitions doivent avoir le même cardinal.)*

### 3 Réponses aux exercices

#### ▷ Question 1, page 1

Supposons qu'il existe un algorithme polynomial  $A$  qui soit une  $\rho$ -approximation pour  $\mathcal{P}$  avec  $\rho < (c+1)/c$ . Soit  $I$  une instance de  $\mathcal{P}$ . Si  $f_{\mathcal{P}}(OPT(I)) \leq c$ , alors  $f_{\mathcal{P}}(A(I)) \leq \rho f_{\mathcal{P}}(OPT(I)) < \frac{c+1}{c} f_{\mathcal{P}}(OPT(I)) < c+1$ , donc  $f_{\mathcal{P}}(A(I)) \leq c$ . Ainsi, l'algorithme fournit une solution de taille inférieure à  $c$  quand l'optimal est de taille inférieure à  $c$  et, dans le cas contraire, fournit nécessairement une solution dont la taille est supérieure strictement à  $c$ . Il peut donc être utilisé pour décider s'il existe une solution de taille inférieure ou égale à  $c$ , ce qui implique que  $P=NP$ .

#### ▷ Question 2, page 1

Si  $D_{opt} < 3w(T_i)$  pour tout  $i$ , alors chaque machine calcule au plus 2 tâches dans l'ordonnancement optimal et on a donc  $n \leq 2p$ . S'il y a  $n = 2p - h$  tâches, un ordonnancement optimal consiste à placer les  $h$  tâches les plus longues seules sur les  $h$  premières machines et à placer les  $2(p-h)$  tâches restantes sur les  $p-h$  machines restantes en les groupant deux par deux (la plus longue avec la plus courte, la seconde plus longue avec la seconde plus courte, et ainsi de suite).

Pour le montrer, considérons un ordonnancement optimal. On a  $2p - h$  tâches à répartir sur  $p$  processeurs avec au plus 2 tâches par processeur. Il y a donc exactement  $h$  tâches seules sur un processeur. On peut toujours échanger ces tâches avec les  $h$  tâches les plus longues sans augmenter la durée de l'ordonnancement. Considérons maintenant deux processeurs responsables chacun de l'exécution de deux tâches  $\{T_{i_1}, T_{i_2}\}$  et  $\{T_{j_1}, T_{j_2}\}$ , où  $T_{i_1}$  est plus longue que  $T_{i_2}$  et  $T_{j_1}$  plus longue que  $T_{j_2}$ . Si  $T_{i_1}$  est plus longue que  $T_{j_1}$  et que  $T_{i_2}$  est plus longue que  $T_{j_2}$ , alors on peut échanger  $T_{i_2}$  et  $T_{j_2}$  sans augmenter la durée de l'ordonnancement. Donc si on ordonne les processeurs de façon à ce que les durées des  $T_{i_1}$  soient croissantes, alors les durées de  $T_{i_2}$  sont décroissantes. Ceci montre que l'ordonnancement décrit est bien optimal.

#### ▷ Question 3, page 1

On va supposer que les tâches sont triées par ordre croissant de durée. Soit  $T_j$  la tâche se terminant en dernier. On a  $D(\sigma) = \tau(T_j) + w(T_j)$ . Aucun des processeurs ne s'arrête donc de travailler avant la date  $\tau(T_j)$ , sinon on aurait assigné cette tâche à un autre processeur. Tous les processeurs ont donc fini de traiter leurs tâches après la date  $\tau(T_j)$ , ce qui signifie que  $\sum_{i \neq j} w(T_i) \geq p \cdot \tau(T_j) = p(D(\sigma) - w(T_j))$ . On a donc  $D(\sigma) \leq \frac{1}{p} \left( \sum_{i \neq j} w(T_i) \right) + w(T_j)$ , ce qui implique que

$$D(\sigma) \leq \frac{1}{p} \sum_i w(T_i) + \left( \frac{p-1}{p} \right) w(T_j) \leq D_{opt} + \left( \frac{p-1}{p} \right) w(T_j).$$

Raisonnons maintenant par l'absurde en supposant que  $D(\sigma) > \left( \frac{4}{3} - \frac{1}{3p} \right) D_{opt}$ . En combinant cette inégalité avec la précédente, on obtient

$$D_{opt} + \left( \frac{p-1}{p} \right) w(T_j) > \left( \frac{4}{3} - \frac{1}{3p} \right) D_{opt}, \text{ soit } D_{opt} < 3w(T_j).$$

Si  $T_j$  est la tâche la plus courte (c'est-à-dire si  $j = n$ ), alors en utilisant le résultat de la question précédente on arrive à une absurdité. En effet, l'algorithme de la question précédente n'est autre que l'algorithme de liste proposé ici et fournit donc un ordonnancement optimal, ce qui contredit l'inégalité  $D(\sigma) > \left( \frac{4}{3} - \frac{1}{3p} \right) D_{opt}$ .

Si ce n'est pas le cas on peut faire le même raisonnement en ne considérant que  $T_1, \dots, T_j$ . En effet, l'ordonnancement  $\sigma'$  défini pour ce sous-ensemble par la procédure de l'énoncé n'est autre que la restriction de  $\sigma$  à ce sous-ensemble. Il a donc la même durée et le  $D_{opt}$  correspondant est plus petit ou égal. L'inégalité n'est donc pas vérifiée non plus pour cet ensemble de tâches. En remarquant enfin que l'ordonnancement fourni par l'algorithme de la question précédente n'est autre que  $\sigma'$ , on en déduit que  $\sigma'$  est donc optimal pour  $T_1, \dots, T_j$ , ce qui est absurde pour la même raison que précédemment.

▷ **Question 4, page 1**

Soit  $G = (V, E)$  un graphe et  $k$  un entier naturel. On doit définir une instance  $I_{(G,k)}$  pour notre problème d'ordonnement tel qu'il existe une clique de taille  $k$  dans  $G$  si et seulement si il existe un ordonnancement de  $I_{(G,k)}$  de durée 3.

Définissons les quantités suivantes :  $\bar{k} = |V| - k$ ,  $l = \frac{1}{2}k(k-1)$  et  $\bar{l} = |E| - l$ . Notre instance sera composée de  $p = \max(k, \bar{k} + l, \bar{l}) + 1$  processeurs et de  $n = 3p$  tâches. À tout sommet  $v$  de  $V$  on associe une tâche  $J_v$ , et à toute arête  $e$  de  $E$ , on associe une tâche  $K_e$ . Les relations entre ces tâches sont les suivantes : si deux sommets  $u$  et  $v$  sont reliés par une arête  $e$  alors  $J_u \prec K_e$  et  $J_v \prec K_e$ . Nous aurons également besoin de tâches factices  $(X_x)_{1 \leq x \leq p-k}$ ,  $(Y_y)_{1 \leq y \leq p-l-\bar{k}}$  et  $(Z_z)_{1 \leq z \leq p-\bar{l}}$ . Clairement, la taille de l'instance ainsi construite est polynomiale en la taille de l'instance initiale.

Montrons qu'il existe une clique de taille  $k$  dans  $G$  si et seulement s'il existe un ordonnancement de  $I_{(G,k)}$  de durée 3.

⇒ Supposons qu'il existe une clique de taille  $k$  dans  $G$ . Alors il est possible de positionner les  $k$  tâches correspondant aux sommets de la clique ainsi que l'intégralité des  $(X_x)_{1 \leq x \leq p-k}$  sur les  $p$  processeurs au temps  $t = [0, 1]$ . On peut ensuite placer au temps  $t = [1, 2]$  les  $l$  tâches correspondant aux arêtes de la clique, les  $\bar{k}$  tâches correspondant aux sommets restants ainsi que l'intégralité des  $(Y_y)_{1 \leq y \leq p-l-\bar{k}}$  tout en respectant les contraintes de dépendances. Enfin, on peut placer au temps  $t = [2, 3]$  toutes les autres tâches, c'est-à-dire les  $\bar{l}$  tâches correspondant aux arêtes n'appartenant pas à la clique et l'intégralité des  $(Z_z)_{1 \leq z \leq p-\bar{l}}$ .

⇐ Supposons qu'il existe un ordonnancement de durée 3. Alors, en raison des dépendances entre les tâches de type  $X$ ,  $Y$ , et  $Z$ , l'intégralité des  $(X_x)_{1 \leq x \leq p-k}$  doit être ordonnée au temps  $t = [0, 1]$ , les tâches  $(Y_y)_{1 \leq y \leq p-l-\bar{k}}$  au temps  $t = [1, 2]$  et les tâches  $(Z_z)_{1 \leq z \leq p-\bar{l}}$  au temps  $t = [2, 3]$ . Étant donné qu'il y a  $p$  processeurs et  $3p$  tâches,  $k$  tâches correspondant à des sommets doivent être ordonnées au temps  $t = [0, 1]$ . Les tâches  $Y$  occupant  $p-l-\bar{k}$  processeurs, on doit trouver  $\bar{k}+l$  tâches parmi les tâches de sommets et d'arêtes ordonnancées au temps  $t = [1, 2]$ . Il y a donc au moins  $l$  tâches associées à des arêtes ordonnancées au temps  $t = [1, 2]$ , ce qui signifie qu'il y a une clique de taille  $k$  dans le graphe  $G$ .

▷ **Question 5, page 1**

En utilisant le théorème 1, on montre qu'il n'existe pas de  $\rho$ -approximation pour  $\rho < 4/3$  si  $P \neq NP$ . Ce type de résultat n'empêche cependant pas l'existence d'un algorithme polynomial fournissant des ordonnancements qui soient, par exemple, proches à une valeur constante de l'optimum<sup>1</sup>. Par contre, si le problème possédait la propriété de passage à l'échelle, on pourrait renforcer notre résultat d'inexistence. Ce n'est malheureusement pas aussi simple qu'il n'y paraît. Les tâches sont unitaires et il est impossible de changer leur durée pour augmenter celle de l'ordonnement. Il faut donc procéder autrement. Soit  $I = (\mathcal{T}, \prec)$  une instance de notre problème d'ordonnement et  $k$  un entier. Construisons une nouvelle instance  $I^{(k)}$  constituée de  $k$  copies  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k$  de  $\mathcal{T}$  ainsi que de  $k-1$  tâches  $J_1, \dots, J_{k-1}$ . Si  $u$  et  $v$  sont dans  $\mathcal{T}_i$  alors  $u \prec^{(k)} v$  si et seulement si  $u \prec v$ . De plus,  $\forall i \forall v \in \mathcal{T}_i : J_{i-1} \prec^{(k)} v \prec^{(k)} J_i$ . Ainsi, s'il existe un ordonnancement de durée  $d$  pour  $I$ , on sait qu'il existe un ordonnancement de durée  $kd + k - 1$  pour  $I^{(k)}$ . De même, d'un ordonnancement de durée  $d$  pour  $I^{(k)}$ , on peut déduire un ordonnancement de durée inférieure à  $(d - (k-1))/k$ . Même si la relation entre  $I$  et  $I^{(k)}$  n'est pas parfaitement linéaire, cette relation est suffisante pour utiliser le théorème ??, ce qui nous permet de renforcer notre résultat d'impossibilité.

▷ **Question 6, page 2**

Soit  $G$  un graphe de FORK  $G$  à  $n$  fils. Commençons par quelques remarques simples concernant un ordonnancement optimal des tâches. La tâche  $T_0$  étant traitée sur le processeur  $P_0$ , il convient de déterminer quelles sont les tâches qui vont être traitées sur  $P_0$  et quelles sont celles qui vont être déléguées à un autre processeur. En effet, si deux tâches sont exécutées sur un processeur  $P_i \neq P_0$ , on n'augmente pas la durée de l'ordonnement en plaçant ces deux tâches sur des processeurs différents. On sait donc qu'il existe un ordonnancement optimal tel qu'un certain ensemble  $\mathcal{I} = \{T_{i_k}, \dots, T_{i_n}\}$  de tâches est calculé

<sup>1</sup>On peut montrer, par exemple, qu'il n'existe pas de  $\rho$ -approximation pour le problème du bin-packing quand  $\rho < 3/2$ . Il existe cependant un algorithme polynomial  $A$  pour ce problème tel que, pour toute instance  $I$ , on ait  $A(I) \leq (11/9) \cdot OPT(I) + 1$

sur le processeur  $P_0$  et tel que les autres tâches sont toutes ordonnancées sur des processeurs distincts. La durée de ce type d'ordonnancement est :

$$T = \max \left( \sum_{i \in \mathcal{I}} w_i, \{w_0 + (d_j + w_j) \mid j \notin \mathcal{I}\} \right).$$

Supposons qu'il existe deux tâches  $T_j \notin \mathcal{I}$  et  $T_k \in \mathcal{I}$  telles que  $w_k + d_k < w_j + d_j$  : alors on n'augmente pas le temps de l'ordonnancement en enlevant la tâche  $T_k$  de  $\mathcal{I}$  et en plaçant celle-ci toute seule sur un autre processeur que le processeur  $P_0$ . On en déduit qu'il existe un ordonnancement optimal tel que toutes les tâches de  $\mathcal{I}$  ont une valeur de  $w_i + d_i$  supérieure à celles n'appartenant pas à  $\mathcal{I}$ .

Pour obtenir un ordonnancement optimal, il suffit alors de trier les tâches  $T_i$  pour  $1 \leq i \leq n$  par  $w_i + d_i$  croissant, et de trouver  $k$  minimisant  $\max(\sum_{i=k}^n w_i, w_{k-1} + d_{k-1})$  : on affectera alors les  $n - k$  dernières tâches à  $P_0$ .

### ▷ Question 7, page 2

La question est facile, on se ramène à 2-Partition de façon similaire à la section ???. Considérons une instance de 2-Partition, c'est-à-dire un ensemble  $\mathcal{A} = \{a_1, \dots, a_n\}$  de  $n$  entiers. Nous allons transformer cette instance en une instance du problème FORK-SCHED-BOUNDED avec deux processeurs qui aura une solution si et seulement si l'instance originale du problème de 2-Partition avait une solution.

Définissons un graphe de FORK  $G$  constitué de  $n + 1$  tâches  $\{T_0, \dots, T_n\}$  :

- le père  $T_0$  a un poids  $w_0 = 0$  ;
- pour tout  $1 \leq i \leq n$  le nœud  $T_i$  a un poids  $w_i = a_i$  ;
- le volume des données de chaque tâche est nul, c'est-à-dire  $d_i = 0$  pour tout  $1 \leq i \leq n$ .

Bien sûr, la taille de l'instance de FORK-SCHED-BOUNDED est linéaire en la taille de l'instance initiale de 2-Partition. Montrons que décider s'il est possible de trouver un ordonnancement du graphe  $G$  en temps inférieur à  $T = \frac{1}{2} \sum_{i=1}^n w_i$  est équivalent à savoir résoudre notre instance de 2-Partition.

⇐ Supposons que l'instance originale de 2-Partition ait une solution : il existe alors  $\mathcal{I}_1$  et  $\mathcal{I}_2$ , deux ensembles partitionnant de  $\{1, \dots, n\}$  telles que  $\sum_{\mathcal{I}_1} a_i = \sum_{\mathcal{I}_2} a_i = S$ . En ordonnanciant  $T_0$  ainsi que les  $T_i$  pour  $i \in \mathcal{I}_1$  sur le premier processeur et les  $T_i$  pour  $i \in \mathcal{I}_2$  sur le second processeur, on obtient un ordonnancement de durée  $\max(\sum_{i \in \mathcal{I}_1} w_i, \sum_{i \in \mathcal{I}_2} w_i) = T$ .

⇒ Supposons l'existence d'un ordonnancement de notre ensemble de tâches en temps  $T$  sur 2 processeurs. Dans ce cas, le processeur  $P_0$  s'occupe d'un ensemble de tâches  $\mathcal{I}_1$  et le processeur  $P_1$  d'un ensemble de tâches  $\mathcal{I}_2$ . On a donc  $\max(\sum_{i \in \mathcal{I}_1} w_i, \sum_{i \in \mathcal{I}_2} w_i) = T$  et  $\sum_{i \in \mathcal{I}_1} w_i + \sum_{i \in \mathcal{I}_2} w_i = 2T$ , ce qui signifie que  $\sum_{i \in \mathcal{I}_1} w_i = \sum_{i \in \mathcal{I}_2} w_i = T$ , c'est-à-dire qu'il existe une solution à notre instance initiale de 2-Partition.

### ▷ Question 8, page 2

Nous réduisons notre problème à 2-Partition-Eq, une variante de 2-Partition dans laquelle les deux sous-ensembles doivent être de même taille. Considérons une instance de 2-Partition-Eq, c'est-à-dire un ensemble  $\mathcal{A} = \{a_1, \dots, a_n\}$  de  $n$  entiers. Nous allons transformer cette instance en une instance du problème FORK-SCHED-1-PORT- $\infty$  qui aura une solution si et seulement si l'instance originale du problème de 2-Partition-Eq en avait une.

Soit  $S = \frac{1}{2} \sum_{i=1}^n a_i$  (si  $S$  n'est pas entier alors il n'y a pas de solution au problème 2-Partition-Eq). Soit  $M = \max a_i$  et  $m = \min a_i$ . Définissons un graphe FORK  $G$  constitué de  $n + 4$  tâches  $\{T_0, \dots, T_{n+3}\}$  :

- le père  $T_0$  a un poids  $w_0 = 0$  ;
- pour tout  $1 \leq i \leq n$  le nœud  $T_i$  a un poids  $w_i = 10(M + a_i + 1)$  ;
- les trois derniers nœuds  $T_{n+1}$ ,  $T_{n+2}$  et  $T_{n+3}$  ont pour poids :  

$$w_{n+1} = w_{n+2} = w_{n+3} = 10(M + m) + 1 ;$$
- le volume des données correspond au volume des calculs, c'est-à-dire  $d_i = w_i$  pour tout  $1 \leq i \leq n + 3$ .

Montrons que décider s'il est possible de trouver un ordonnancement du graphe  $G$  en temps inférieur à  $T = \frac{1}{2} \sum_{i=1}^n w_i + 2w_{n+1} = 5n(M + 1) + 10S + 20(M + m) + 2$  est équivalent à savoir résoudre notre instance de 2-Partition-Eq.

⇐ Supposons que l'instance originale de 2-Partition ait une solution : il existe alors  $\mathcal{I}_1$  et  $\mathcal{I}_2$ , deux sous-ensembles partitionnant  $\{1, \dots, n\}$  tels que  $\sum_{\mathcal{I}_1} a_i = \sum_{\mathcal{I}_2} a_i = S$ . Construisons notre ordonnancement de la façon suivante :

- Le processeur  $P_0$  est en charge de l'exécution de la tâche  $T_0$  et des tâches  $T_i$  pour  $i \in \mathcal{I}_1$  et des tâches  $T_{n+1}$  et  $T_{n+2}$ .  $P_0$  a besoin exactement de  $T$  unités de temps pour effectuer l'ensemble de ses calculs puisque  $T = \frac{1}{2} \sum_{i=1}^n w_i + 2w_{n+1}$  et que  $\frac{1}{2} \sum_{i=1}^n w_i = \sum_{i \in \mathcal{I}_1} w_i$ .
- Chaque tâche restante est assignée à un processeur différent. Nous utilisons donc  $|\mathcal{I}_2| + 1$  processeurs en plus de  $P_0$ .
- Les communications sont faites suivant l'ordre croissant des indices des tâches : ainsi le dernier message envoyé concerne la tâche  $T_{n+3}$ .
- Le processeur responsable de  $T_{n+3}$  est donc prêt à commencer son exécution à l'instant  $\sum_{\mathcal{I}_2} d_i + d_{n+3}$  et termine son exécution à l'instant

$$\sum_{\mathcal{I}_2} d_i + d_{n+3} + w_{n+3} = T$$

- Tous les autres processeurs terminent leur exécution plus tôt. En effet, ils reçoivent leur message au plus tard en  $\sum_{\mathcal{I}_2} d_i$  et leur durée exécution  $w_i$  est inférieure à  $2w_{n+3}$ .
- Nous avons donc construit un ordonnancement valide de notre instance de FORK-SCHED-1-PORT- $\infty$ .

$\Rightarrow$  Réciproquement, supposons que notre instance de FORK-SCHED-1-PORT- $\infty$  possède une solution, c'est-à-dire un ordonnancement  $\sigma$  qui permet d'obtenir un temps d'exécution inférieur à  $T$ . Notons  $P_0$  le processeur qui exécute la tâche  $T_0$  et  $\mathcal{I} = \{i \mid 1 \leq i \leq n+3 \text{ et } T_i \text{ est traitée sur } P_0\}$  l'ensemble des indices des tâches affectées à  $P_0$ . Le temps de calcul de  $P_0$  est au moins de  $A = \sum_{i \in \mathcal{I}} w_i$ . Le processeur qui reçoit le dernier message de  $P_0$  pour exécuter la tâche  $T_{\text{last}}$  (dont l'indice n'est pas dans  $\mathcal{I}$ ) ne peut pas terminer son exécution avant  $B = \sum_{i \notin \mathcal{I}} d_i + w_{\text{last}}$ . Comme l'ordonnancement  $\sigma$  donne une solution à notre instance de FORK-SCHED-1-PORT- $\infty$ , nous avons  $\max(A, B) \leq T$ . Or  $A + B = \sum_i w_i + w_{\text{last}} = 2T + w_{\text{last}} - w_{n+1}$  est supérieur ou égal à  $2T$ . Nécessairement,  $A = B = T$  et  $w_{n+1} = w_{\text{last}}$ . Comme  $A = B$ , nous avons en particulier  $A \equiv B[10]$ . Donc  $\mathcal{I}$  contient au moins deux indices parmi  $\{n+1, n+2, n+3\}$ . En considérant  $\mathcal{I}_1$  égal à  $\mathcal{I}$  privé de ces deux indices et  $\mathcal{I}_2 = \{1, \dots, n\} \setminus \mathcal{I}_1$ , on obtient une solution de l'instance initiale de 2-Partition-Eq.

Clairement, la taille de l'instance de FORK-SCHED-1-PORT- $\infty$  est linéaire en la taille de l'instance de 2-Partition-Égales, ce qui achève la preuve de NP-complétude.