

# Ordonnancement avec communications

**Résumé:** Dans ce TD, nous continuons notre tour d’horizon des problèmes d’ordonnancement mais en considérant cette fois-ci les communications. Nous démontrons des résultats de complexité (solution polynomiale ou démonstration de NP-complétude) pour chacun de ces problèmes.

## 1 Ordonnancement d’un graphe FORK hétérogène

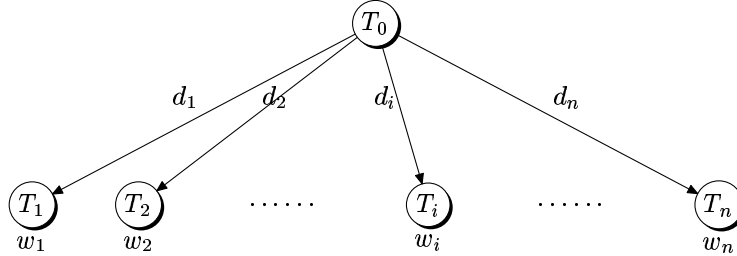


FIG. 1 – graphe de FORK à  $N$  fils

Définissons la notion de graphe de FORK à  $n$  fils :

*Définition 1 (FORK à  $n$  fils).* Un graphe de FORK à  $n$  fils est graphe de tâches à  $n + 1$  noeuds étiquetés par  $T_0, T_1, \dots, T_n$  comme illustré sur la figure 1. Il y a une arête entre le noeud  $T_0$  et chacun de ses fils  $T_i$ ,  $1 \leq i \leq n$ . Chaque noeud 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.

Dans cette section, nous supposons disposer d’une architecture avec un certain nombre de processeurs de même vitesse et d’un réseau de communication lui aussi homogène.

### 1.1 Avec une infinité de processeurs identiques : multi-port

Nous supposons ici que l’on dispose d’un nombre illimité de machines multi-ports, c’est à dire qui peuvent à la fois calculer et communiquer avec n’importe lesquelles des autres machines.

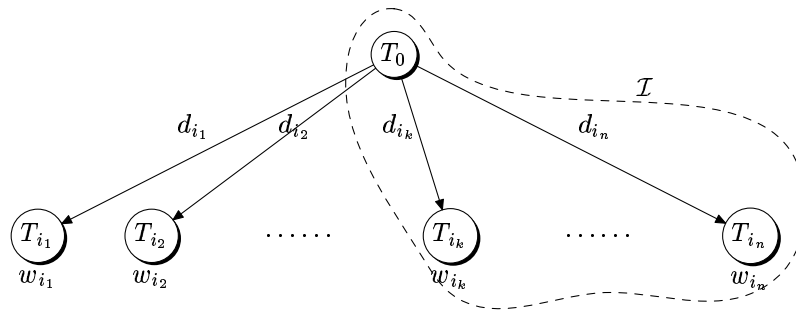
Étant donné un graphe de FORK et une telle architecture, nous définissons le problème d’optimisation suivant :

*Définition 2 (FORK-SCHED-MPORT- $\infty(G)$ ).* Étant donné un graphe de FORK  $G$  à  $n$  fils et un ensemble infini de processeurs multi-ports de même vitesse connectés par un réseau homogène, quel est la durée de l’ordonnancement  $\sigma$  qui minimise le temps d’exécution ?

► **Question 1.** Donner un algorithme polynomial résolvant FORK-SCHED-MPORT- $\infty$ .

*Réponse.* Soit  $G$  un graphe de FORK  $G$  à  $n$  fils.

Commençons par quelques banalités 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 le processeur  $P_0$  et quelles sont celles qui vont être déléguée 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’ordonnancement 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 sur le processeur  $P_0$  et tel que les autres tâches sont toutes ordonnancées sur des processeurs distincts les uns des autres.



La durée de ce type d'ordonnancement est

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

Notons  $T_{i_j}$  la tâche n'appartenant pas à  $\mathcal{I}$  et de  $d_i + w_i$  maximum. S'il existe une tâche  $T_{i_l} \notin \mathcal{I}$  (c'est à dire telle que  $k \leq l$ ) telle que  $w_{i_j} + d_{i_j} > w_{i_l} + d_{i_l}$ , alors il est possible de diminuer la durée de cet ordonnancement en plaçant la tâche  $T_{i_k}$  toute seule sur un autre processeur que le processeur  $P_0$ . On en déduit donc que dans un ordonnancement optimal 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 donc de trier les tâches  $T_i$  pour  $1 \leq i \leq n$  par  $w_i + d_i$  croissant et de trouver  $k$  minimisant  $\max(\sum_{i=k}^n w_i, w_{k-1} + d_{k-1})$ .  $\square$

## 1.2 Avec un nombre fini de processeurs identiques : multi-port

On s'intéresse ici au même problème que précédemment mais à ressource limitée. On ne dispose plus d'une infinité de processeurs mais d'un nombre borné de processeurs.

*Définition 3 (FORK-SCHED-MPORT-BOUNDED( $G, p$ )).* Étant donné un graphe de FORK  $G$  à  $N$  fils et un ensemble de  $p$  processeurs multi-ports de même vitesse connectés par un réseau homogène, quel est la durée de l'ordonnancement  $\sigma$  qui minimise le temps d'exécution ?

▷ **Question 2.** Montrer que le problème de décision associé à FORK-SCHED-MPORT-BOUNDED est NP-complet à partir du moment où il y a plusieurs processeurs.

*Réponse.* Nous allons nous ramener à 2-partition. 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-MPORT-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 noeud  $T_i$  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  $1 \leq i \leq n$ .

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 partitions de  $\{1, \dots, n\}$  tels que  $\sum_{i \in \mathcal{I}_1} a_i = \sum_{i \in \mathcal{I}_2} a_i = S$ . En ordonnant  $T_0$  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 processeurs, 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.  $\square$

### 1.3 Avec une infinité de processeurs identiques : un port

Dans cette partie, nous allons établir la NP-complétude du problème d'ordonnancement dans le modèle 1-port (une machine peut calculer et communiquer en même temps mais avec un seul processeur à la fois).

*Définition 4 (FORK-SCHED-1-PORT- $\infty(G)$ ).* Étant donné un graphe de FORK  $G$  à  $N$  fils et un ensemble infini de processeurs 1-port de même vitesse connectés par un réseau homogène, quel est la durée de l'ordonnancement  $\sigma$  qui minimise le temps d'exécution ?

▷ **Question 3.** Démontrer que le problème de décision associé à FORK-SCHED-1-PORT- $\infty$  est NP-complet. (*Indication : on pourra se ramener à 2-Partition*)

*Réponse.* Nous commençons par considérer 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-1-PORT- $\infty$  qui aura une solution si et seulement si l'instance originale du problème de 2-Partition avait une solution.

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). Soit  $M = \max a_i$  et  $m = \min a_i$ . Définissons un graphe de 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 noeud  $T_i$  a un poids  $w_i = 10(M + a_i + 1)$ ,
- les trois derniers noeuds  $T_{n+1}, T_{n+2}$  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.

⇐ Supposons que l'instance originale de 2-Partition ait une solution : il existe alors  $\mathcal{I}_1$  et  $\mathcal{I}_2$ , deux partitions de  $\{1, \dots, n\}$  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  $T = \frac{1}{2} \sum_{i=1}^n w_i + 2w_{n+1}$ .
- Chaque tâche restante est assigné 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 exécution  $w_i$  est inférieur à  $2w_{n+3}$ .

Nous avons donc construit un ordonnancement valide de notre instance de FORK-SCHED-1-PORT- $\infty$ .

⇒ 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  $A + B = \sum_i w_i + w_{\text{last}} = 2T + w_{\text{last}} - w_{n+1} \geq 2T$ . 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 donc une solution de 2-Partition pour notre instance initiale.

□

## 2 Ordonnancement d'un ensemble de tâches identiques sur une plateforme maître-esclave hétérogène.

Dans cette section, on dispose d'un ensemble de tâches identiques et indépendantes ainsi que d'une plateforme maître-esclave constituée de processeurs de vitesses différentes (voir Figure 2). Le modèle de machine utilisé est celui des machines 1-port, c'est à dire qu'il n'est pas possible de communiquer et de calculer en même temps.

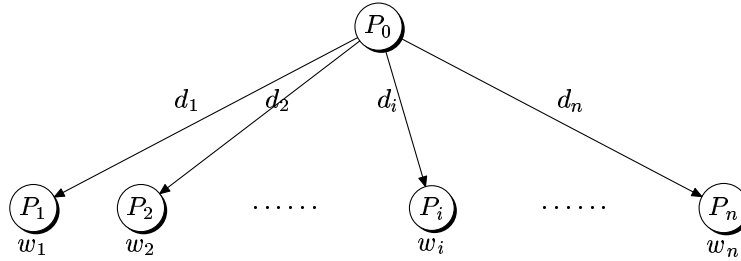


FIG. 2 – plateforme maître-esclave hétérogène

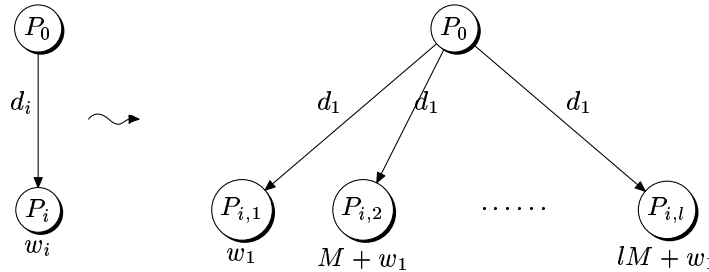
Le calcul d'une tâche sur le processeur  $P_i$  requière une communication de durée  $d_i$  avec  $P_0$  et dure  $w_i$ .

*Définition 5 (MASTER-SLAVE( $(d_1, w_1), \dots, (d_n, w_n), T$ )).* Étant donné un ensemble de tâches identiques et indépendantes et une plateforme maître-esclaves de processeurs multi-ports de caractéristiques  $(d_1, w_1), \dots, (d_n, w_n)$ , quel est le nombre maximal de tâches que l'on puisse calculer en temps  $T$  ?

On va montrer que ce problème peut être résolu en temps polynomial.

▷ **Question 4.** Étant donné une instance  $((d_1, w_1), \dots, (d_n, w_n), T)$  de MASTER-SLAVE, créer une nouvelle instance telle que chaque processeur exécute au plus une tâche.

*Réponse.* Il suffit de transformer le processeur  $i$  de la façon suivante :



avec  $M = \max(d_i, w_i)$  et  $l = \lfloor \frac{T-w_i}{M} \rfloor$ . □

▷ **Question 5.** Soit  $(P_{i_1}, \dots, P_{i_k})$  une liste de processeur telle que l'on puisse déléguer une tâche à  $P_{i_1}$  puis à  $P_{i_2}$ , ..., puis à  $P_{i_k}$  en un temps inférieur à  $T$ . Montrer que l'on peut également déléguer ces tâches en prenant ces processeurs dans l'ordre des  $w_i$  décroissants.

*Réponse.* Réordonnons les processeurs dans l'ordre des  $w_i$  décroissants et notons donc  $i_1, \dots, i_k$  l'ordre qui marche. On a donc :

$$\begin{array}{rcl} (L_1) & d_{i_1} + w_{i_1} & \leq T \\ (L_2) & d_{i_1} + d_{i_2} + w_{i_2} & \leq T \\ & \vdots & \vdots \\ (L_k) & d_{i_1} + d_{i_2} + \dots + d_{i_k} + w_{i_k} & \leq T \end{array}$$

Soient  $j_1, \dots, j_k$  tels que  $i_{j_1} = 1, i_{j_2} = 2, \dots$  et  $i_{j_k} = k$ . Démontrons par récurrence que les équations induites par l'ordre des  $w_i$  décroissants sont vérifiées.

- i) On a  $d_1 + w_1 \leq T$ . En effet, dans l'équation  $L_{j_1}$  on a  $d_{i_1} + \dots + d_{i_{j_1-1}} + d_1 + w_1 \leq T$ .
- ii) On a  $d_1 + d_2 + w_2 \leq T$ . En effet, il y a deux possibilités :
- soit  $j_2 > j_1$  et alors l'inégalité  $L_{j_2}$  comprend les termes  $d_{i_{j_1}} + d_{i_{j_2}} + w_{i_{j_2}}$  et donc on a bien  $d_1 + d_2 + w_2 \leq T$ ,
  - soit  $j_2 < j_1$  et alors l'inégalité  $L_{j_1}$  comprend les termes  $d_{i_{j_1}} + d_{i_{j_2}} + w_{i_{j_1}}$  et, en utilisant le fait que  $w_2 \leq w_1$ , on a bien  $d_1 + d_2 + w_2 \leq T$ .
- iii) Passons au cas général. Soit  $i < k$ . Supposons que pour tout  $l \leq i$  on ait  $d_1 + \dots + d_l + w_l \leq T$  et montrons que  $d_1 + \dots + d_{i+1} + w_{i+1} \leq T$ . Notons  $m_i = \max(j_1, \dots, j_i)$ .
- Soit  $j_{i+1} > m_i$  et on regarde directement l'équation  $L_{j_{i+1}}$ . En effet cette équation contient les termes  $d_1, d_2, \dots, d_i$  (car  $j_{i+1} > m_i$ ) ainsi que  $d_{i+1} + w_{i+1}$  et donc on a bien  $d_1 + \dots + d_{i+1} + w_{i+1} \leq T$ .
  - Soit  $j_{i+1} < m_i$  et on regarde l'équation  $L_{m_i}$ . En effet cette équation contient les termes  $d_1, d_2, \dots, d_{i+1}$  ainsi que  $d_i + w_i$ . On en déduit donc que  $d_1 + \dots + d_{i+1} + w_i \leq T$ . En utilisant le fait que  $w_i \geq w_{i+1}$ , on obtient donc bien  $d_1 + \dots + d_{i+1} + w_{i+1} \leq T$ .

□

On propose l'algorithme suivant :

```

MASTER-SLAVE((P1, ..., Pn), T)
1:  Trier les processeurs par di croissant
2:  L ← ∅
3:  Pour i = 1 à n
4:    Si L ∪ {Pi} est ordonnançable en temps inférieur à T
5:    Alors L ← L ∪ {Pi}
6:  Renvoyer (L)

```

▷ **Question 6.** Supposons qu'il existe une liste de processeurs  $(P_{i_1}, \dots, P_{i_k})$  qui marche en temps  $T$  dans cet ordre. Montrer que l'algorithme glouton précédent renvoie au moins  $n$  valeurs.

*Réponse.* Pour alléger les notations, on dira qu'un ensemble d'indices  $\mathcal{I} = \{i_1, \dots, i_k\}$  est ordonnançable si et seulement s'il existe un ordonnancement valide en temps inférieur à  $T$  des processeurs  $\{P_{i_1}, \dots, P_{i_k}\}$ .

On va montrer que l'algorithme renvoie des valeurs  $\{g_1, \dots, g_k\}$  et que cet ensemble est bien ordonnançable. On supposera dans la suite que l'ensemble  $\{i_1, \dots, i_k\}$  est trié par  $c_i$  croissant.

- i) On sait que l'ensemble des indices  $\mathcal{I} = \{i_1, \dots, i_k\}$  est ordonnançable et donc que chaque  $\{i_j\}$  est ordonnançable. Ceci nous garantit donc l'existence de  $g_1$  et que  $g_1 \leq \min(i_1, \dots, i_k)$ .  
Si  $g_1 \in \mathcal{I}$  il existe bien une solution à  $k$  éléments contenant  $g_1$ . Dans le cas contraire,  $\mathcal{I}$  étant ordonnançable dans un certain ordre, en remplaçant le premier indice  $i_{j_1}$  de  $\mathcal{I}$  à être ordonné par  $g_1$ , on obtient toujours un ensemble d'équations consistant : la validité de la première équation ( $c_{g_1} + w_{g_1} \leq T$ ) découle de la définition de  $g_1$  et la validité des autres équations du fait que l'on a remplacé  $c_{i_{j_1}}$  par  $c_{g_1}$  qui est plus petit.
- ii) On sait qu'il existe un ensemble  $\mathcal{I} = \{g_1, i'_1, \dots, i'_{k-1}\}$  ordonnançable. Donc pour tout  $j$ , on a  $\{g_1, i'_j\}$  ordonnançable, ce qui nous garantit l'existence de  $g_2$  ainsi que le fait que  $g_2 \leq \min(i'_1, \dots, i'_{k-1})$ .  
Si  $g_2 \in \{i'_1, \dots, i'_{k-1}\}$  alors, on sait qu'il existe un ensemble à  $k$  éléments contenant  $g_1$  et  $g_2$ . Dans le cas contraire, on va remplacer la première inégalité qui ne concerne pas  $g_1$  par l'inégalité induite par  $g_2$  :
- $g_1$  est l'indice du premier processeur qui reçoit une tâche et  $f$  est l'indice du deuxième. Étant donné que  $\{g_1, g_2\}$  est ordonnançable, en remplaçant  $f$  par  $g_2$  dans  $\mathcal{I}$ , on satisfait toujours les deux premières inégalités. Les autres découlent du fait que  $c_{g_1} \leq c_f$ .
  - L'indice  $f$  du premier processeur qui reçoit une tâche n'est pas  $g_1$ . Dans ce cas, en remplaçant  $f$  par  $g_2$  dans  $\mathcal{I}$ , la première inégalité est clairement vérifiée ( $\{g_2\}$  est ordonnançable) et les suivantes découlent du fait que  $c_{g_1} \leq c_f$ .

Il existe donc bien une solution à  $k$  éléments contenant  $g_1$  et  $g_2$ .

- iii) Passons au cas général. Supposons qu'il existe un ensemble  $\mathcal{I} = \{g_1, \dots, g_{j-1}, i'_1, \dots, i'_{k-j+1}\}$  ordonnançable. Étant donné que pour tout  $l \in \llbracket 1, k-j+1 \rrbracket : \{g_1, \dots, g_{j-1}, i'_l\}$  est ordonnançable, on est assuré de l'existence de  $g_j$  et que  $g_j \leq \min(i'_1, \dots, i'_{k-j+1})$ .

Si  $g_j \in \{i'_1, \dots, i'_{k-j+1}\}$  alors on est assuré de l'existence d'un ensemble à  $k$  éléments ordonnançable et contenant  $g_1, \dots, g_{j-1}$  et  $g_j$ . Dans le cas contraire, on va remplacer la première inégalité  $L_f$  qui ne concerne pas  $g_1, g_2, \dots$  ou  $g_{j-1}$  par l'inégalité induite par  $g_j$ .

Si la première inégalité concerne  $g_{i_1}$ , la deuxième  $g_{i_2}, \dots$ , la  $l^{\text{ème}}$   $g_{i_l}$  et la  $l+1^{\text{ème}}$   $i'_f$ , alors i)  $\{g_{i_1}, \dots, g_{i_l}, i'_f\}$  (en temps que sous ensemble de  $\mathcal{I}$ ) et ii)  $\{g_{i_1}, \dots, g_{i_l}, g_j\}$  (en temps que sous ensemble de  $\{g_1, \dots, g_j\}$ ) sont ordonnançables. Quand on remplace  $i'_f$  par  $g_j$  dans  $\mathcal{I}$ , les  $l$  premières équations sont inchangées, la  $l+1^{\text{ème}}$  découle de ii), et les suivantes du fait que  $c_{g_j} \leq c_{i'_f}$ .

□