Optimisation

Ordonnancement de graphes

Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

Master 2 Informatique – Semestre 9

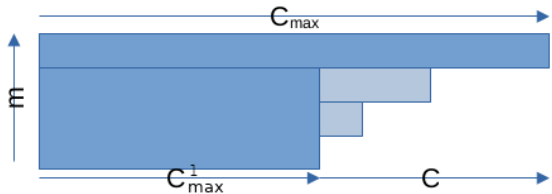
Plan

Plan

LST : rappel (1/2)

Rappel de cours : LST est un algorithme d'approximation $(2 - \frac{1}{m})$ (borne de Graham). Résultat basé sur 2 bornes inférieures sur le makespan optimal OPT.

- Temps d'exécution maximum : $\text{OPT} \geq \max_j p_j$.
- Charge moyenne : $\text{OPT} \geq \frac{1}{m} \sum_j p_j$.



Preuve simplifiée :

- Pas de délai avant $C^1_{\max}$: $m \times C^1_{\max} \leq \sum_j p_j$ et donc $C^1_{\max} \leq \text{OPT}$.
- Pas de tâche commençant après $C^1_{\max}$: $C \leq \max_j p_j$ et donc $C \leq \text{OPT}$.
- Intégration : $\text{LST} = C^1_{\max} + C \leq 2 \times \text{OPT}$.

LST : $Q||C_{\max}$ (2/2)

Extension possible pour le problème $Q||C_{\max}$:

- Le temps d'exécution de la tâche T_j dépend du temps de cycle de la machine i : $p_{ij} = c_i \times p_j$.
- On peut adapter la preuve et on obtient une borne similaire.

LPT : preuve pour le facteur $\frac{4}{3}$ (1/4)

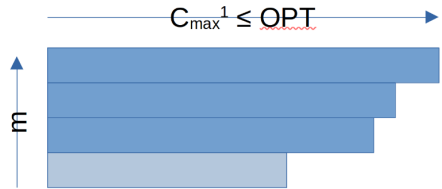
La preuve suivante est donnée à titre illustratif et pour l'ouverture :

- Preuve directe (sans contradiction) et simplifiée proposé par Xiao Xin en 2017.
- 3 types de tâches : courtes ($p_j \leq \frac{\text{OPT}}{3}$), médianes ($\frac{\text{OPT}}{3} < p_j \leq \frac{2\text{OPT}}{3}$) et longues ($\frac{2\text{OPT}}{3} < p_j$).
- Une seule tâche longue par machine dans OPT : il y a au plus $n_L = m$ longues tâches.
- Pas de tâche médiane après une tâche longue et pas plus de 2 tâches médianes par machine dans OPT : il y a au plus $2(m - n_L)$ tâches médianes.
- 3 phases finissant à 3 dates : $C^1_{\max}$, $C^2_{\max}$ et $C^3_{\max}$.

LPT : preuve pour le facteur $\frac{4}{3}$ (2/4)

Phase 1 : LPT ordonnance les m premières tâches (les plus longues), chacune sur une machine distincte.
Résultat à l'issue de cette phase :

- Toutes les tâches longues sont ordonnancées.
- $C^1_{max} \leq \text{OPT}$.

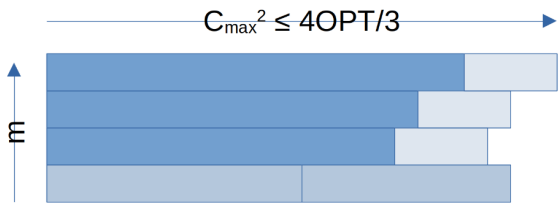


LPT : preuve pour le facteur $\frac{4}{3}$ $(3/4)$

Phase 2 : LPT ordonnance les tâches suivantes en s'arrêtant juste avant de mettre une troisième tâche sur une machine.

Résultat à l'issue de cette phase :

- Il y a au plus 2 tâches sur chaque machine.
- Toutes les tâches médianes sont ordonnancées.
- Si une tâche est rajoutée après une tâches longue, c'est forcément une courte.
- $C_{max}^2 \leq \frac{4OPT}{3}$.

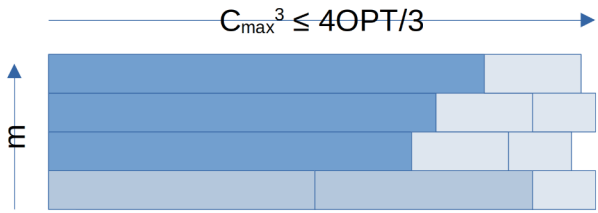


LPT : preuve pour le facteur $\frac{4}{3}$ $(4/4)$

Phase 3 : LPT finit de traiter les dernières tâches.

Résultat à l'issue de cette phase :

- Toutes les tâches courtes sont ordonnancées.
- La dernière tâche qui termine est précédée :
  - soit de 0 ou 1 tâche (cas déjà traité) ;
  - soit d'au moins 2 tâches : c'est donc une petite tâche et elle débute avant ou en même temps que OPT car toutes les machines sont alors occupées.
- $C_{max} = C^3_{max} \leq \frac{4OPT}{3}$.



MULTIFIT : bornes initiales (1/2)

Bornes initiales de la recherche dichotomique sur le makespan :

- inférieure (aucun ordonnancement possible en dessous) : $L = \left\lceil \max(\frac{1}{m}\sum_j p_j, \max_j p_j) \right\rceil$ (charge moyenne et temps d'exécution maximum) ;
- supérieure (FFD produit forcément une solution valide) : $U = \left\lceil \max(\frac{2}{m}\sum_j p_j, \max_j p_j) \right\rceil$.

Preuve pour la borne supérieure :

- Supposons que FFD échoue avec un makespan U.
- On considère la première tâche T_j qui ne peut pas être ordonnancée.
- On déduit que $j > m$ car chaque tâche $T_{j'} \leq m$ peut être ordonnancée sur sa propre machine.
- Soit $p_j > \frac{U}{2}$. Toutes les m première tâches sont plus longues et donc on a déjà au moins ordonnancé $\sum_{j=1}^{m} p_j > m\frac{U}{2} \geq \sum_j p_j$. C'est une contradiction.
- Soit $p_j \leq \frac{U}{2}$. Si on a pas pu l'ordonnancer, c'est que chaque machine est trop occupée. La somme de leurs temps d'exécution dépasse $m\frac{U}{2}$. C'est la même contradiction.

MULTIFIT : facteur d'approximation 2 (2/2)

Observations préliminaires :

- Cela garantit que FFD renvoi toujours un ordonnancement lorsque le makespan considéré est $2 \times \text{OPT}$.
- Si FFD échouait, une quantité de travail conséquente aurait été exécutée.
- Les bornes sur le makespan optimal restent pertinentes.

SLACK : approximation ?

SLACK est un algorithme récent (Della Croce et Scatamacchia, 2020) basé sur la stratégie gloutonne suivante :

- Trier les tâches par ordre décroissant de leur taille ;
- Découper l'ensemble trié en tuples de m tâches ;
- Soit "slack" la différence entre la taille du premier job et la taille du dernier job de chaque tuple ;
- Trier l'ensemble des tuples dans l'ordre décroissant de leur "slack".

Question ouverte : peut-on prouver un facteur d'approximation pour SLACK (même large) ?

SLACK : approximation ?

SLACK est un algorithme récent (Della Croce et Scatamacchia, 2020) basé sur la stratégie gloutonne suivante :

- Trier les tâches par ordre décroissant de leur taille ;
- Découper l'ensemble trié en tuples de m tâches ;
- Soit "slack" la différence entre la taille du premier job et la taille du dernier job de chaque tuple ;
- Trier l'ensemble des tuples dans l'ordre décroissant de leur "slack".

Question ouverte : peut-on prouver un facteur d'approximation pour SLACK (même large) ?

Exemple de résultats de recherche issus du laboratoire FEMTO-ST associé à la formation (Canon, Dugois, Héam, Jecker, 2025) : facteur d'approximation 4/3 optimal.

Plan

Technique de preuve

- Un argument d'échange consiste à montrer par l'absurde qu'une solution qui ne respecte pas une règle donnée (SPT, LPT, etc.) peut être améliorée en échangeant 2 tâches.
- Technique utilisée pour montrer que SPT est optimal pour le problème $1||\sum C_j$.

$1||\max w_j C_j$

- Le problème consiste à minimiser le maximum pondéré des temps de terminaison.
- La stratégie optimale consiste à prioriser les tâches avec une forte pondération (exécuter les tâches par ordre décroissant de pondération).
- On le prouve en considérant qu'une solution meilleure existe avec 2 tâches, T_1 et T_2 telles que $C_1 < C_2$ et $w_1 < w_2$.
- Comme $w_1 C_1 < w_2 C_2$, la valeur de l'objectif obtenue est $w_2 C_2$.
- Après échange des tâches les temps de terminaison sont notées $C'_1 = C_2$ et $C'_2 = C_1$.
- Ainsi, $w_1 C'_1 = w_1 C_2 < w_2 C_2$ et $w_2 C'_2 = w_2 C_1 < w_2 C_2$ donc l'objectif est au moins aussi bon après échange.
- Il suffit de répéter ces échanges qui n'empirent pas l'objectif jusqu'à obtenir l'ordre décroissant de pondération avec un objectif au moins aussi bon.
- C'est une contradiction car on avait supposé que cette autre solution était meilleure.

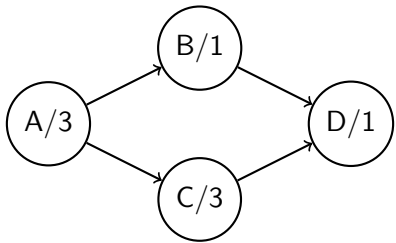
$P|p_j = p| \max w_j C_j$

- Le problème consiste à minimiser le maximum pondéré des temps de terminaison sur plusieurs machines avec des temps d'exécution identiques.
- La stratégie optimale consiste encore à prioriser les tâches avec une forte pondération.
- La preuve consiste également à réaliser un échange entre 2 tâches qui ne respectent pas ce principe.

Plan

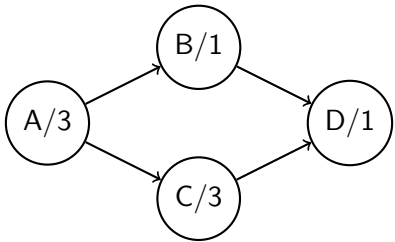
Contraintes de dépendance

- Chaque tâche devient disponible dès que tous ses ancêtres sont terminés.
- Les dépendances sont renseignées dans un graphe orienté sans circuit (*Directed Acyclic Graph*).
- On peut supposer qu'il y a toujours une source et un puits.
- On rajoute "prec" dans le champ β de la notation $\alpha|\beta|\gamma$.
- Exemple : la tâche D ne peut être exécutée que lorsque B et C terminent.



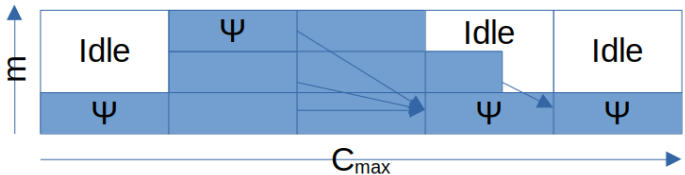
Bottom level

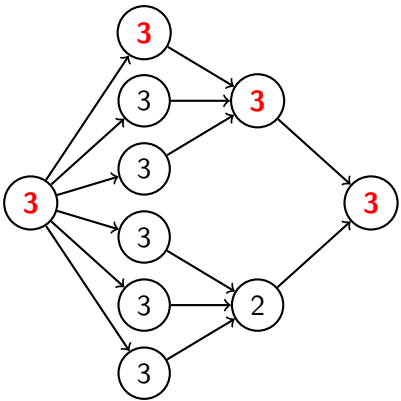
- Le *bottom level* d'une tâche est la longueur maximale entre cette tâche et le puits (en incluant son temps d'exécution).
- C'est le temps minimal pour finir l'exécution du graphe s'il reste au moins cette tâche à faire.
- Le *bottom level* de la source est une borne inférieure du makespan optimal.
- Exemple : on a 1 pour D, 4 pour C, 2 pour B et 7 pour A.



Borne de Graham pour $P|prec|C_{max}$ (1/2)

- Extension de la borne de Graham au cas avec dépendances.
- Soit Ψ un *chemin critique* : ensemble de tâches sur un chemin (entre la source et le puits) et qui ne peuvent être rallongées sans augmenter le makespan.
- Résultat décisif : Idle $\leq (m-1)\sum_{j\in\Psi} p_j$.





Borne de Graham pour $P|prec|C_{\max}$ (2/2)

- Rappel du slide précédent : Idle $\leq (m-1)\sum_{j\in\Psi} p_j$.
- Adaptation des 2 bornes de base :
  - Charge moyenne (identique au cas indépendant) : OPT $\geq \frac{1}{m}\sum_j p_j$.
  - Temps d'exécution maximum (sur le chemin critique Ψ) : OPT $\geq \sum_{j\in\Psi} p_j$.
- L'intégration se fait avec $C_{\max} = \frac{\sum_j p_j + \text{Idle}}{m}$.
- On a donc :

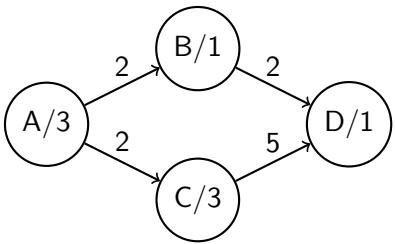
$$C_{\max} \leq \text{OPT} + \left(1 - \frac{1}{m}\right)\sum_{j\in\Psi} p_j \leq \left(2 - \frac{1}{m}\right)\text{OPT}$$

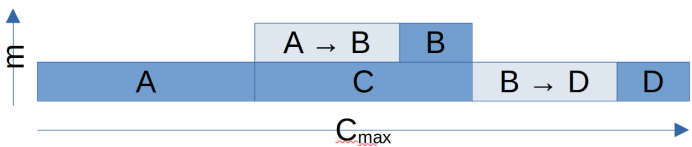
Coûts de communication

- Un temps de transfert peut être associée à chaque dépendance (communication de données).
- Si un transfert a lieu, la machine destinatrice devient occupée jusqu'à ce qu'elle exécute la tâche fille.
- On ignore la contention dans le modèle de communication : les temps de transfert sont indépendants de ce qu'il se passe sur le réseau.
- Le temps de transfert devient nul si la tâche fille est exécutée sur la même machine que la tâche mère.

Critical Path Scheduling

- L'algorithme *Critical Path Scheduling* tri les tâches par *bottom level* décroissant.
- Il procède ensuite comme un algorithme de liste : à chaque pas de temps, la première tâche disponible est ordonnancée sur la première machine disponible.





Plan

Résumé

- De nombreux facteurs d'approximation existent pour les heuristiques LST, LPT, MULTIFIT et SLACK.
- On peut prouver certains résultats en montrant qu'on ne peut pas empirer une solution en échangeant 2 tâches.
- On peut représenter des dépendances entres les tâches dans un graphe de tâches. Les résultats en indépendant peuvent parfois se généraliser aux graphes.

Prochaines échéances

- Second rendu intermédiaire du projet-tournoi le 20/10 (avant la prochaine séance)
- Épreuve sur table le 23/10.
- Rendu final pour le projet-tournoi le 4/11.
- Épreuve de TP le 5/11.
- Restitution du projet-tournoi le 6/11.