M1Info - Optimisation et Recherche Opérationnelle

Cours 4-Ordonnancement

Semestre Automne 2018-2019 - Université Lyon 1

département
Informatique

Faculté des Sciences et Technologies
Université Claude Bernard Lyon 1

Formulation générale

- **Entrée** :
  - nombre de machines m
  - nombre de tâches n
  - caractéristiques des tâches : leur durée d_i, précédences éventuelles, dates d'arrivée, ...
- **Solution** : Pour chaque tâche le moment où elle est exécutée et la machine qui l'éxécute.
- **Objectif :** Suivant le problème : minimiser le temps d'éxécution total, le nombre de machines utilisées, les tâches en retard, la somme des temps d'éxécution,...

Exemple : Ordonnancement de processus, planification de projets, gestion de file d'attente,...

Un premier exemple

- 2 machines
- 6 tâches non sécables, avec leur durée

Tache	A	B	C	D	E	F
Durée	3	4	7	2	1	4

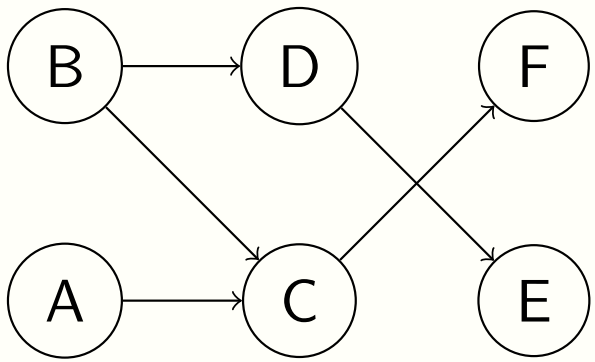
- **Objectif** Minimiser le temps final.

Un premier exemple

- 2 machines
- 6 tâches non sécables, avec leur durée

Tache	A	B	C	D	E	F
Durée	3	4	7	2	1	4

- **Objectif** Minimiser le temps final.
  - Possible en 11 et optimal (car la somme des temps est 21).
- Et si on ajoute des précédences ?

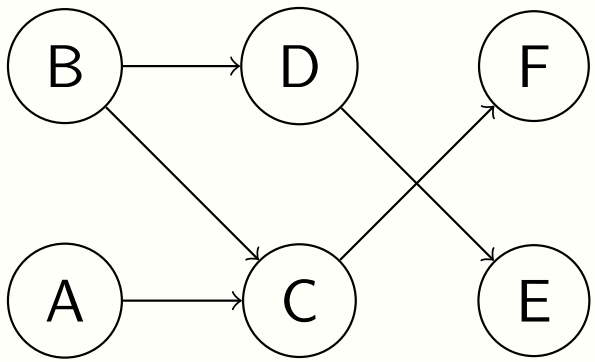


Un premier exemple

- 2 machines
- 6 tâches non sécables, avec leur durée

Tache	A	B	C	D	E	F
Durée	3	4	7	2	1	4

- **Objectif** Minimiser le temps final.
  - Possible en 11 et optimal (car la somme des temps est 21).
- Et si on ajoute des précédences ?



  - Possible et optimal en 15.

Différentes résolutions

- Idée naïve : tester tous les ordres possibles
  - Ordre de grandeur ?

Différentes résolutions

- Idée naïve : tester tous les ordres possibles
  - Ordre de grandeur ? Exponentiel !
    Pour n tâches, il y a $n!$ ordres possible. Avec $n = 20$, cela fait 10^{16} ordres à faire. Si un ordre testé en $12\mu s$, cela prend 1927 ans...

Différentes résolutions

- Idée naïve : tester tous les ordres possibles
  - Ordre de grandeur ? Exponentiel !
    Pour n tâches, il y a $n!$ ordres possible. Avec $n = 20$, cela fait 10^{16} ordres à faire. Si un ordre testé en $12\mu s$, cela prend 1927 ans...
  - Mais donnera la solution exacte
- Méthodes polynomiales. Plusieurs possibilités :
  - Résolution exacte ($\rightarrow$ problème dans P)
  - Résolution approchée, avec garantie (*)
  - Résolution approchée, sans garantie

(*) Un algorithme de minimisation est une k-approximation si pour toute instance I la solution renvoyée par l'algorithme est au plus k fois la solution optimale :

$$Sol_{algo}(I) \leq k \times Sol_{optimale}(I)$$

Exemple : il existe une 2-approximation polynomiale pour le voyageur de commerce euclidien

Algorithme de liste

Une méthode assez fréquente :

- On ordonne les jobs dans une liste suivant un critère (par exemple, par durée décroissante)
- Lorsqu'une machine est dispo, on lui donne à faire le premier job dans la liste qui est possible.

Algorithme de liste

Une méthode assez fréquente :

- On ordonne les jobs dans une liste suivant un critère (par exemple, par durée décroissante)
- Lorsqu'une machine est dispo, on lui donne à faire le premier job dans la liste qui est possible.

Sur l'exemple précédent, avec la liste par durée décroissante

Tache	A	B	C	D	E	F
Durée	3	4	7	2	1	4

Liste : $L = \{C, B, F, A, D, E\}$

Algorithme de liste

Une méthode assez fréquente :

- On ordonne les jobs dans une liste suivant un critère (par exemple, par durée décroissante)
- Lorsqu'une machine est dispo, on lui donne à faire le premier job dans la liste qui est possible.

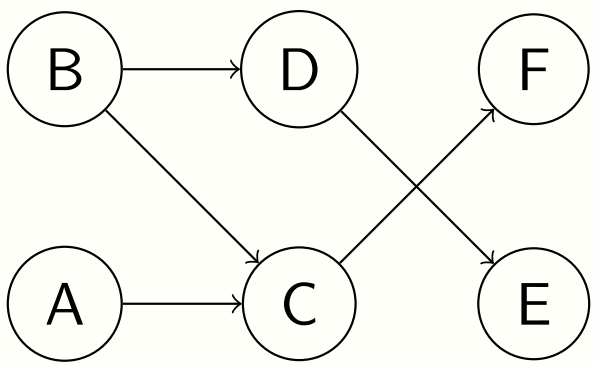
Sur l'exemple précédent, avec la liste par durée décroissante

Tache	A	B	C	D	E	F
Durée	3	4	7	2	1	4

Liste : $L = \{C, B, F, A, D, E\}$

Sans précédence, on obtient une durée de 11

Avec précédence



Algorithme de liste

Une méthode assez fréquente :

- On ordonne les jobs dans une liste suivant un critère (par exemple, par durée décroissante)
- Lorsqu'une machine est dispo, on lui donne à faire le premier job dans la liste qui est possible.

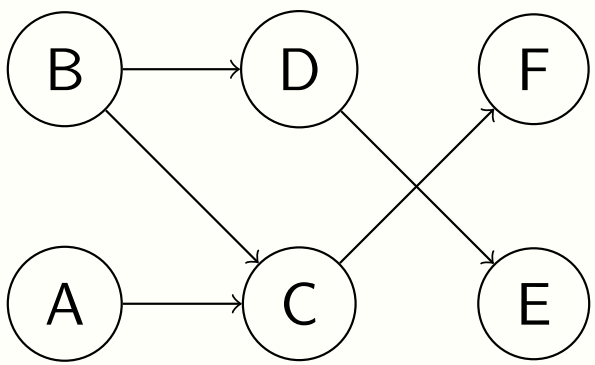
Sur l'exemple précédent, avec la liste par durée décroissante

Tache	A	B	C	D	E	F
Durée	3	4	7	2	1	4

Liste : $L = \{C, B, F, A, D, E\}$

Sans précédence, on obtient une durée de 11

Avec précédence on obtient bien 15



Anomalies de Graham

Entrées : 9 tâches avec précédences.

On veut minimiser le temps final.

Algorithme de liste avec la liste $\{A, B, C, D, ...\}$.

Job	Durée	Précédences
A	3	
B	2	
C	2	
D	2	
E	4	D
F	4	D
G	4	D
H	4	D
I	9	A

- Avec 3 machines :
- Avec 4 machines :

Anomalies de Graham

Entrées : 9 tâches avec précédences.

On veut minimiser le temps final.

Algorithme de liste avec la liste $\{A, B, C, D, ...\}$.

Job	Durée	Précédences
A	3	
B	2	
C	2	
D	2	
E	4	D
F	4	D
G	4	D
H	4	D
I	9	A

- Avec 3 machines :

  12 unités de temps, optimal

- Avec 4 machines :

Anomalies de Graham

Entrées : 9 tâches avec précédences.
On veut minimiser le temps final.
Algorithme de liste avec la liste $\{A, B, C, D, ...\}$.

Job	Durée	Précédences
A	3	
B	2	
C	2	
D	2	
E	4	D
F	4	D
G	4	D
H	4	D
I	9	A

- Avec 3 machines :
  12 unités de temps, optimal
- Avec 4 machines :
  15 unités de temps, pas optimal et plus long qu'avec 3 !

Anomalies de Graham

Entrées : 9 tâches avec précédences.
On veut minimiser le temps final.
Algorithme de liste avec la liste $\{A, B, C, D, ...\}$.

Job	Durée	Précédences
A	2	
B	1	
C	1	
D	1	
E	3	D
F	3	D
G	3	D
H	3	D
I	8	A

- Avec 3 machines :
  12 unités de temps, optimal
- Avec 4 machines :
  15 unités de temps, pas optimal et plus long qu'avec 3 !
- Avec 3 machines, mais tous les jobs durent -1 :

Anomalies de Graham

Entrées : 9 tâches avec précédences.
On veut minimiser le temps final.
Algorithme de liste avec la liste $\{A, B, C, D, ...\}$.

Job	Durée	Précédences
A	2	
B	1	
C	1	
D	1	
E	3	D
F	3	D
G	3	D
H	3	D
I	8	A

- Avec 3 machines :
  12 unités de temps, optimal
- Avec 4 machines :
  15 unités de temps, pas optimal et plus long qu'avec 3 !
- Avec 3 machines, mais tous les jobs durent -1 :
  13 unités de temps, pas optimal et moins bien que 1er cas !

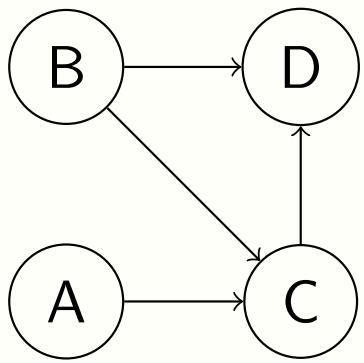
Quelques exemples de résolutions

- Nombre de machines : 1, 2, m, illimité
- Pas de précédences/ précédences
- Objectif : minimiser temps final/ somme des temps finaux

1 machine, minimiser le temps final

- Sans précédence :
  - n'importe quel ordre donne le même temps optimal
- Avec précédences :

Job	Durée
A	3
B	4
C	1
D	2

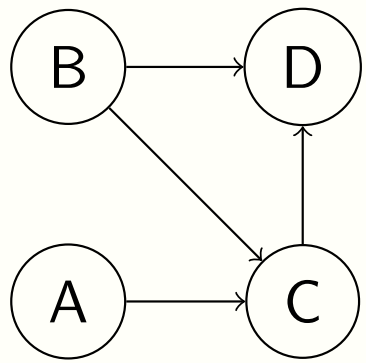


Résolution exacte en temps polynomial.

1 machine, minimiser le temps final

- Sans précédence :
  - n'importe quel ordre donne le même temps optimal
- Avec précédences :
  - Ordre topologique : prendre un sommet sans précédence et le retirer puis recommencer.

Job	Durée
A	3
B	4
C	1
D	2



Résolution exacte en temps polynomial.

1 machine, minimiser la somme des temps finaux

Exemple : file d'attente $\Leftrightarrow$ min le temps attendu

Job	Durée
A	3
B	4
C	1
D	2

- Si tout le monde arrive en même temps :

1 machine, minimiser la somme des temps finaux

Exemple : file d'attente $\Leftrightarrow$ min le temps attendu

Job	Durée
A	3
B	4
C	1
D	2

- Si tout le monde arrive en même temps :
  - Algo de liste par durée croissante

1 machine, minimiser la somme des temps finaux

Exemple : file d'attente $\Leftrightarrow$ min le temps attendu

Job	Durée
A	3
B	4
C	1
D	2

- Si tout le monde arrive en même temps :
  - Algo de liste par durée croissante
  - Solution optimale en temps $n \log n$, preuve avec argument d'échange

1 machine, minimiser la somme des temps finaux

Exemple : file d'attente $\Leftrightarrow$ min le temps attendu

Job	Durée	Arrivée
A	3	5
B	4	0
C	1	1
D	2	2

- Si tout le monde arrive en même temps :
  - Algo de liste par durée croissante
  - Solution optimale en temps $n \log n$, preuve avec argument d'échange
- Si pas tous en meme temps :

1 machine, minimiser la somme des temps finaux

Exemple : file d'attente $\Leftrightarrow$ min le temps attendu

Job	Durée	Arrivée
A	3	5
B	4	0
C	1	1
D	2	2

- Si tout le monde arrive en même temps :
  - Algo de liste par durée croissante
  - Solution optimale en temps $n \log n$, preuve avec argument d'échange
- Si pas tous en meme temps :
  - Algo de liste par durée croissante ?

1 machine, minimiser la somme des temps finaux

Exemple : file d'attente $\Leftrightarrow$ min le temps attendu

Job	Durée	Arrivée
A	3	5
B	4	0
C	1	1
D	2	2

- Si tout le monde arrive en même temps :
  - Algo de liste par durée croissante
  - Solution optimale en temps $n \log n$, preuve avec argument d'échange
- Si pas tous en meme temps :
  - Algo de liste par durée croissante ? pas optimal
  - NP complet mais 2 approx (en TD)

Machines illimitées, précédences, minimiser temps final

Exemple : construction d'une maison

Job	Durée	précédences
A	3	
B	2	A
C	4	A
D	3	B,C
E	2	B
F	4	D,E

On construit le graphe potentiel-tâche et on peut déterminer :

- La durée minimale du projet et un chemin critique
- Les dates au plus tôt et au plus tard de chaque tâche (pour faire le projet dans le temps min)

m machines, minimiser le temps final

Pas de précédences

- $m = 2$: problème équivalent à Sac-à-dos (cf TD1)
  - NP-complet, programmation dynamique,...
- m quelconque ?
  - Toujours NP-complet
  - Mais n'importe quel algo de liste donne une $(2 - 1/m)$-approximation. (*)
  - En prenant plus gros en premier on obtient une $(4/3 - 1/3m)$-approx.

Exemple : 3 machines avec

Job	Durée
A	4
B	2
C	3
D	5
E	6
F	1

m machines, minimiser le temps final

Pas de précédences

- $m = 2$: problème équivalent à Sac-à-dos (cf TD1)
  - NP-complet, programmation dynamique,...
- m quelconque ?
  - Toujours NP-complet
  - Mais n'importe quel algo de liste donne une $(2 - 1/m)$-approximation. (*)
  - En prenant plus gros en premier on obtient une $(4/3 - 1/3m)$-approx.

Exemple : 3 machines avec

Job	Durée
A	4
B	2
C	3
D	5
E	6
F	1

Preuve de (*) au slide suivant

Preuve du facteur $2 - 1/m$ (Slide 1/3)

- On considère un algo de liste quelconque et une instance quelconque I. On veut démontrer que

$$Sol_{algo}(I) \leq (2 - \frac{1}{m}) \times Sol_{opt}(I)$$

- On considère la tâche qui finit en dernier, sans perte de généralités, on peut supposer que c'est la tâche n.
- Soit t_n la date de début de la tâche n. On a donc

$$sol_{algo}(I) = t_n + d_n$$

- Avant la date t_n, toutes les machines sont occupées à tout instant (principe de l'algo de liste - il reste au moins une tâche, la tâche n dans la liste donc une machine ne reste jamais inactive avant t_n).

Preuve du facteur $2 - 1/m$ (Slide 2/3)

- Donc la durée total des tâches à faire, $\sum_{i=1}^{n} d_i$, est au moins égale à $m \times t_n$ (ce qui est fait avant la date t_n) plus la durée de la tâche n. Soit :

$$\sum_{i=1}^{n} d_i \geq m t_n + d_n$$

- La solution optimale est au moins la durée totale de ce qu'il y a à faire divisé par le nombre de machines :

$$sol_{opt}(I) \geq \sum_{i=1}^{n} d_i/m$$

- En utilisant l'inégalité précédente, on a donc :

$$sol_{opt}(I) \geq t_n + d_n/m$$

- En retournant l'inégalité :

$$t_n \leq sol_{opt}(I) - d_n/m$$

Preuve du facteur $2 - 1/m$ (Slide 3/3)

- En mettant cette inégalité dans l'expression de la solution :

$$sol_{algo}(I) = t_n + d_n \leq sol_{opt}(I) + d_n(1 - 1/m)$$

- Dans la solution optimale, on a une machine qui doit faire la tâche n en entier, on a donc forcément aussi :

$$d_n \leq sol_{opt}(I)$$

- On peut conclure :

$$sol_{algo}(I) \leq sol_{opt}(I) + sol_{opt}(I)(1 - 1/m) = (2 - 1/m)sol_{opt}(I)$$