

# Bases de l'ordonnancement

## Problèmes d'ordonnancement à machines parallèles

Safia Kedad-Sidhoum

CNAM  
safia.kedad\_sidhoum@cnam.fr

BOR - M2 MPRO, 2020-2021

# Introduction

- Ces problèmes apparaissent naturellement dans les systèmes informatiques où l'on dispose de processeurs parallèles
- Il existe trois types de problèmes :
  - $P$  machines **identiques**
  - $Q$  machines **uniformes**: une vitesse est associée à chaque machines
  - $R$  machines **non reliées**: la durée de la tâche dépend de la machine (sans rapport de proportionalité)

# Introduction

- Ces problèmes apparaissent naturellement dans les systèmes informatiques où l'on dispose de processeurs parallèles
- Il existe trois types de problèmes :
  - $P$  machines **identiques**
  - $Q$  machines **uniformes**: une vitesse est associée à chaque machines
  - $R$  machines **non reliées**: la durée de la tâche dépend de la machine (sans rapport de proportionalité)

# Machines identiques

## Définition et notations

- $n$  tâches  $1, 2, \dots, n$  non préemptives de durées  $p_1, p_2, \dots, p_n$
- $m$  machines identiques  $M_1, M_2, \dots, M_m$
- chaque tâche requiert une machine pour son exécution
- chaque machine ne peut exécuter au plus qu'une tâche à la fois

# Machines identiques

## Définition et notations

- $n$  tâches  $1, 2, \dots, n$  non préemptives de durées  $p_1, p_2, \dots, p_n$
- $m$  machines **identiques**  $M_1, M_2, \dots, M_m$
- chaque tâche requiert une machine pour son exécution
- chaque machine ne peut exécuter au plus qu'une tâche à la fois

# Machines identiques

## Définition et notations

- $n$  tâches  $1, 2, \dots, n$  non préemptives de durées  $p_1, p_2, \dots, p_n$
- $m$  machines **identiques**  $M_1, M_2, \dots, M_m$
- chaque tâche requiert une machine pour son exécution
- chaque machine ne peut exécuter au plus qu'une tâche à la fois

# Machines identiques

## Définition et notations

- $n$  tâches  $1, 2, \dots, n$  non préemptives de durées  $p_1, p_2, \dots, p_n$
- $m$  machines **identiques**  $M_1, M_2, \dots, M_m$
- chaque tâche requiert une machine pour son exécution
- chaque machine ne peut exécuter au plus qu'une tâche à la fois

$$P| - |\sum C_i$$

- **Complexité:** Problème polynomial.
  - On suppose que  $n = km$  (en ajoutant si besoin des tâches de durées nulles)
  - Dominance des ordonnancements dit **équilibrés**: chaque machine exécute exactement  $k$  tâches
  - Ordonnement optimal: allouer de façon circulaire les tâches aux  $m$  machines dans un ordre non décroissant de leur durée



$$P| - |\sum C_i$$

- **Complexité:** Problème polynomial.
  - On suppose que  $n = km$  (en ajoutant si besoin des tâches de durées nulles)
  - Dominance des ordonnancements dit **équilibrés**: chaque machine exécute exactement  $k$  tâches
  - Ordonnement **optimal**: allouer de façon circulaire les tâches aux  $m$  machines dans un **ordre non décroissant de leur durée**

$$P| - |\sum C_i$$

- **Complexité:** Problème polynomial.
  - On suppose que  $n = km$  (en ajoutant si besoin des tâches de durées nulles)
  - Dominance des ordonnancements dit **équilibrés**: chaque machine exécute exactement  $k$  tâches
  - Ordonnement **optimal**: allouer de façon circulaire les tâches aux  $m$  machines dans un **ordre non décroissant de leur durée**

$$P| - |\sum C_i$$

- **Complexité:** Problème polynomial.
  - On suppose que  $n = km$  (en ajoutant si besoin des tâches de durées nulles)
  - Dominance des ordonnancements dit **équilibrés**: chaque machine exécute exactement  $k$  tâches
  - Ordonnement **optimal**: allouer de façon circulaire les tâches aux  $m$  machines dans un **ordre non décroissant de leur durée**

$$P \mid - \mid C_{\max}$$

- Règle SPT: Que peut-on dire de l'ordonnement de liste SPT ?
- Complexité: NP-difficile au sens fort

$$P | - | C_{\max}$$

- Règle SPT: Que peut-on dire de l'ordonnement de liste SPT ?
- Complexité: NP-difficile au sens fort

# Contraintes de précedence

- Critère: minimiser le makespan.
- On suppose de plus que les tâches doivent satisfaire des relations de précedence définies par un graphe de précedence
- Problème:  $P|prec|C_{max}$
- Complexité: Problème NP-difficile au sens fort.

# Contraintes de précedence

- Critère: minimiser le makespan.
- On suppose de plus que les tâches doivent satisfaire des relations de précedence définies par un graphe de précedence
- Problème:  $P|prec|C_{max}$
- Complexité: Problème NP-difficile au sens fort.

# Contraintes de précedence

- Critère: minimiser le makespan.
- On suppose de plus que les tâches doivent satisfaire des relations de précedence définies par un graphe de précedence
- Problème:  $P|prec|C_{max}$
- Complexité: Problème NP-difficile au sens fort.



# Contraintes de précedence

- Critère: minimiser le makespan.
- On suppose de plus que les tâches doivent satisfaire des relations de précedence définies par un graphe de précedence
- Problème:  $P|prec|C_{max}$
- Complexité: Problème NP-difficile au sens fort.

## Contraintes de précedence - suite

- Tout algorithme de liste fournit une  $2 - \frac{1}{m}$  approximation
- Preuve au tableau
- Amélioration: règle LPT fournit une  $\frac{4}{3} - \frac{1}{3m}$  approximation

## Contraintes de précedence - suite

- Tout algorithme de liste fournit une  $2 - \frac{1}{m}$  approximation
- Preuve au tableau
- Amélioration: règle LPT fournit une  $\frac{4}{3} - \frac{1}{3m}$  approximation

# Contraintes de précedence - Cas polynomiaux

$P|_{intree, p_i = 1} | C_{max}$

- Le graphe de précedence  $G$  est une **anti-arborescence**.
- Les durées des tâches sont **unitaires**.
- Soit  $n_j$  le niveau de la tâche dans  $G$ , c'est-à-dire le nombre de sommets du chemin de  $j$  à la racine de  $G$ .
- Soit  $L = (1, 2, \dots, n)$  est une liste de tâches ordonnées par niveau décroissant au sens large.
- Algorithme de Hu: Ordonnement de liste basé sur  $L$  est un ordonnement de makespan minimal.
- Exemple au tableau.

# Contraintes de précédence - Cas polynomiaux

$P |_{intree, p_i = 1} | C_{\max}$

- Le graphe de précédence  $G$  est une **anti-arborescence**.
- Les durées des tâches sont **unitaires**.
- Soit  $n_j$  le niveau de la tâche dans  $G$ , c'est-à-dire le nombre de sommets du chemin de  $j$  à la racine de  $G$ .
- Soit  $L = (1, 2, \dots, n)$  est une liste de tâches ordonnées par **niveau décroissant** au sens large.
- **Algorithme de Hu**: Ordonnement de liste basé sur  $L$  est un ordonnancement de makespan minimal.
- Exemple au tableau.

# Contraintes de précédence - Cas polynomiaux

$P|_{intree, p_i = 1} C_{\max}$

- Le graphe de précédence  $G$  est une **anti-arborescence**.
- Les durées des tâches sont **unitaires**.
- Soit  $n_j$  le niveau de la tâche dans  $G$ , c'est-à-dire le nombre de sommets du chemin de  $j$  à la racine de  $G$ .
- Soit  $L = (1, 2, \dots, n)$  est une liste de tâches ordonnées par **niveau décroissant** au sens large.
- **Algorithme de Hu**: Ordonnement de liste basé sur  $L$  est un ordonnancement de makespan minimal.
- Exemple au tableau.

# Contraintes de précedence - Cas polynomiaux

$P|_{intree, p_i = 1} | C_{max}$

- Le graphe de précedence  $G$  est une **anti-arborescence**.
- Les durées des tâches sont **unitaires**.
- Soit  $n_j$  le niveau de la tâche dans  $G$ , c'est-à-dire le nombre de sommets du chemin de  $j$  à la racine de  $G$ .
- Soit  $L = (1, 2, \dots, n)$  est une liste de tâches ordonnées par **niveau décroissant** au sens large.
- **Algorithme de Hu**: Ordonnement de liste basé sur  $L$  est un ordonnement de makespan minimal.
- Exemple au tableau.

# Contraintes de précedence - Cas polynomiaux

$P|_{intree, p_i = 1} C_{max}$

- Le graphe de précedence  $G$  est une **anti-arborescence**.
- Les durées des tâches sont **unitaires**.
- Soit  $n_j$  le niveau de la tâche dans  $G$ , c'est-à-dire le nombre de sommets du chemin de  $j$  à la racine de  $G$ .
- Soit  $L = (1, 2, \dots, n)$  est une liste de tâches ordonnées par **niveau décroissant** au sens large.
- **Algorithme de Hu**: Ordonnement de liste basé sur  $L$  est un ordonnement de makespan minimal.
- Exemple au tableau.



# Contraintes de précedence - Cas polynomiaux

$P|_{intree, p_i = 1} | C_{max}$

- Le graphe de précedence  $G$  est une **anti-arborescence**.
- Les durées des tâches sont **unitaires**.
- Soit  $n_j$  le niveau de la tâche dans  $G$ , c'est-à-dire le nombre de sommets du chemin de  $j$  à la racine de  $G$ .
- Soit  $L = (1, 2, \dots, n)$  est une liste de tâches ordonnées par **niveau décroissant** au sens large.
- **Algorithme de Hu**: Ordonnement de liste basé sur  $L$  est un ordonnement de makespan minimal.
- Exemple au tableau.

# Contraintes de précédence - Cas polynomiaux

$P2|prec, p_i = 1|C_{\max}$

- **Algorithme de Coffman-Graham**: algorithme de liste efficace  $O(n^2)$  pour résoudre ce problème.
- Soit  $n_i$  est la longueur en nombre de sommets d'un plus long chemin de  $G$  d'origine  $i$ .
- Idées de base :
  - si  $n_i > n_j$  alors  $i$  doit être plus prioritaire que  $j$
  - si  $n_i = n_j$  et  $Succ(i) \subset Succ(j)$  alors  $j$  doit être plus prioritaire que  $i$ .

# Contraintes de précédence - Cas polynomiaux

$P2|prec, p_i = 1|C_{\max}$

- **Algorithme de Coffman-Graham**: algorithme de liste efficace  $O(n^2)$  pour résoudre ce problème.
- Soit  $n_i$  est la longueur en nombre de sommets d'un plus long chemin de  $G$  d'origine  $i$ .
- Idées de base :
  - si  $n_i > n_j$  alors  $i$  doit être plus prioritaire que  $j$
  - si  $n_i = n_j$  et  $Succ(i) \subset Succ(j)$  alors  $j$  doit être plus prioritaire que  $i$ .

# Autres cas polynomiaux

- Durées unitaires:
  - $P|p_i = 1, r_i, d_i|-$ : Problème de flots
- Prémption
  - $P|pmtn|C_{\max}$ : Algorithme de Mac-Naughton
  - $P|pmtn, r_i, d_i|-$ : Problème de flots