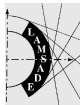


Introduction aux problèmes d'ordonnancement

Mohamed Ali ALOULOU

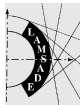
LAMSADE
Université Paris Dauphine
E-mail : aloulou@lamsade.dauphine.fr

28 novembre 2005



Plan du cours

- 1 Définition et formulation du problème d'ordonnancement
- 2 Ordonnancement de projet : rappels et extensions
- 3 Ordonnancement d'ateliers : contexte et classification
- 4 Ordonnancement d'ateliers : méthodes de résolution



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

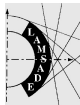
Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Partie 1 : Définition et formulation du problème d'ordonnancement

- 1 C'est quoi l'ordonnancement ?
- 2 Quelques domaines concernés par la fonction ordonnancement
- 3 La fonction ordonnancement dans la gestion de la production
- 4 Contraintes rencontrées en ordonnancement
 - Contraintes de potentiel
 - Contraintes disjonctives
 - Contraintes cumulatives
- 5 Formulation mathématique



Plan

C'est quoi
l'ordonnancement ?

Domaines

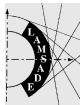
Ordonnancement en
GdPContraintes en
ordonnancementContraintes de potentiel
Contraintes disjonctives
Contraintes cumulativesFormulation
mathématique

Ordonnancer ?

Définition

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînements, ...) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises.

- Un ensemble de **tâches**
 - Un environnement de **ressources** pour effectuer les tâches
 - Des **contraintes** sur les tâches et les ressources
 - Un **critère d'optimisation**
- ⇒ Déterminer les dates d'exécution des tâches



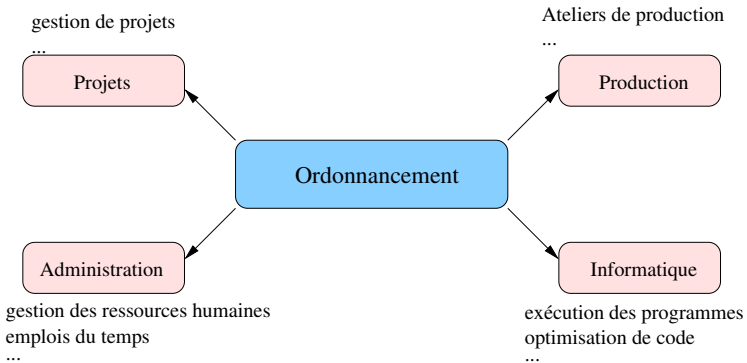
Plan

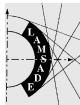
C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdPContraintes en
ordonnancementContraintes de potentiel
Contraintes disjonctives
Contraintes cumulativesFormulation
mathématique

Domaines concernés





Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement
Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

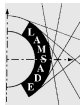
Formulation
mathématique

La gestion de la production [Giard 2003]

La gestion de production a pour objet la recherche d'une organisation efficace de la production des biens et des services

3 catégories pour classer les décisions en gestion de la production :

- les décisions **stratégiques** : politique long terme de l'entreprise
- les décisions **tactiques** : décisions à moyen terme
 - planification de la production
 - plan de transport
- les décisions **opérationnelles** : court terme
 - gestion des stocks
 - ordonnancement
 - pilotage informatique en temps réel



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

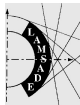
Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Contraintes rencontrées en ordonnancement

Différentes contraintes

- **technologiques** : une tâche ne peut débuter que lorsque d'autres sont achevées
- **commerciales** : certaines dates doivent être achevées pour une date fixée
- **matérielles** : une machine ne peut traiter qu'une machine à la fois
- **de main d'oeuvre** : effectif limité
- **financières** : budget limité



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

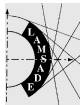
Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Contraintes rencontrées en ordonnancement

Différentes contraintes

- technologiques : une tâche ne peut débuter que lorsque d'autres sont achevées
- commerciales : certaines dates doivent être achevées pour une date fixée
- matérielles : une machine ne peut traiter qu'une machine à la fois
- de main d'oeuvre : effectif limité
- financières : budget limité



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

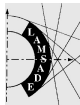
Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Contraintes rencontrées en ordonnancement

Différentes contraintes

- technologiques : une tâche ne peut débuter que lorsque d'autres sont achevées
- commerciales : certaines dates doivent être achevées pour une date fixée
- matérielles : une machine ne peut traiter qu'une machine à la fois
- de main d'oeuvre : effectif limité
- financières : budget limité



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

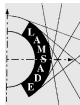
Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Contraintes rencontrées en ordonnancement

Différentes contraintes

- technologiques : une tâche ne peut débuter que lorsque d'autres sont achevées
- commerciales : certaines dates doivent être achevées pour une date fixée
- matérielles : une machine ne peut traiter qu'une machine à la fois
- de main d'oeuvre : effectif limité
- financières : budget limité



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

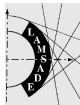
Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Contraintes rencontrées en ordonnancement

Différentes contraintes

- technologiques : une tâche ne peut débuter que lorsque d'autres sont achevées
- commerciales : certaines dates doivent être achevées pour une date fixée
- matérielles : une machine ne peut traiter qu'une machine à la fois
- de main d'oeuvre : effectif limité
- financières : budget limité



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

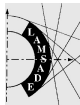
Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Formalisation des contraintes

- 1 Contraintes potentielles (ou de potentiels)
- 2 Contraintes disjonctives
- 3 Contraintes cumulatives



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

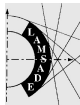
Formulation
mathématique

Contraintes potentielles

Notation : t_j : date de début de la tâche j , p_j sa durée

Forme générale : $t_j - t_i \geq a_{ij}$

- Localisation temporelle : j ne peut débuter avant une certaine date (livraison de matière première, conditions climatiques,...)
- Contrainte de délai : j doit être terminée avant une certaine date
- Contrainte de succession



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

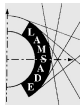
Formulation
mathématique

Contraintes potentielles

Notation : t_j : date de début de la tâche j , p_j sa durée

Forme générale : $t_j - t_i \geq a_{ij}$

- Localisation temporelle : j ne peut débuter avant une certaine date (livraison de matière première, conditions climatiques,...)
- Contrainte de délai : j doit être terminée avant une certaine date
- Contrainte de succession
 - succession simple
 - succession avec attente
 - succession avec chevauchement
 - succession immédiate



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

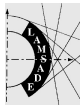
Formulation
mathématique

Contraintes potentielles

Notation : t_j : date de début de la tâche j , p_j sa durée

Forme générale : $t_j - t_i \geq a_{ij}$

- Localisation temporelle : j ne peut débuter avant une certaine date (livraison de matière première, conditions climatiques,...)
- Contrainte de délai : j doit être terminée avant une certaine date
- Contrainte de succession
 - succession simple
 - succession avec attente
 - succession avec chevauchement
 - succession immédiate



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

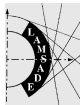
Formulation
mathématique

Contraintes potentielles

Notation : t_j : date de début de la tâche j , p_j sa durée

Forme générale : $t_j - t_i \geq a_{ij}$

- Localisation temporelle : j ne peut débuter avant une certaine date (livraison de matière première, conditions climatiques,...)
- Contrainte de délai : j doit être terminée avant une certaine date
- Contrainte de succession
 - succession simple
 - succession avec attente
 - succession avec chevauchement
 - succession immédiate



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

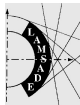
Contraintes disjonctives

Deux tâches i et j sont en disjonction si elles ne peuvent être exécutées simultanément

⇒ Les intervalles d'exécution des tâches disjonctives sont disjoints : $]t_i, t_i + p_i[\cap]t_j, t_j + p_j[= \emptyset$

Disjonction d'inégalités de potentiels

$$t_j - t_i \geq p_i \quad \text{ou} \quad t_i - t_j \geq p_j$$



Plan

C'est quoi
l'ordonnement ?

Domaines

Ordonnement en
GdP

Contraintes en
ordonnement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Contraintes cumulatives

C'est une généralisation des contraintes disjonctives

Exemple : On a deux grues et 5 tâches nécessitant une grue sont candidates au même moment

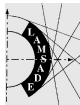
Soit

- $w_k(t)$ la quantité de moyen k disponible à t .
- $w_{ik}(t)$ la quantité de moyen k nécessaire pour exécuter i à t .

Si

$$\sum_{i \in S} w_{ik}(t) \leq w_k(t)$$

alors les tâches de S peuvent être exécutées simultanément à t
sinon les tâches de S sont en disjonction



Plan

C'est quoi
l'ordonnancement ?

Domaines

Ordonnancement en
GdP

Contraintes en
ordonnancement

Contraintes de potentiel
Contraintes disjonctives
Contraintes cumulatives

Formulation
mathématique

Formulation mathématique

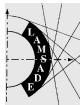
n tâches à exécuter + 2 tâches fictives 0 et $n + 1$ de durées nulles.

Déterminer $(t_0, t_1, \dots, t_n, t_{n+1})$ de façon à
Minimiser $f(t_0, t_1, \dots, t_n, t_{n+1})$

s.c.

- 1 Contraintes de potentiel : $t_j - t_i \geq a_{ij}$
- 2 Contraintes disjonctives : $t_j - t_i \geq p_i$ ou $t_i - t_j \geq p_j$
- 3 Contraintes cumulatives (idem)
- 4 Contraintes de non négativité : $t_0, t_1, \dots, t_n, t_{n+1} \geq 0$

f est fonction des dates de début (ou de fin des tâches), par exemple, $f(t_0, t_1, \dots, t_n, t_{n+1}) = t_{n+1} - t_0$



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Partie 2 : Ordonnancement de projet – rappels et extensions

- 6** Problème central de l'ordonnancement : ressources illimitées
 - Définition
 - Modélisation avec un graphe potentiels-tâches
 - Recherche d'ordonnancement admissible
- 7** Cas général : ressources limitées
 - Problématique
 - Résolution exacte
 - Approche simple de résolution : algorithme de liste
- 8** Cas de ressources financières
 - Problématique
 - L'offre et la demande
 - Algorithme de décalage



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition

Modélisation

Résolution

Cas général : ressources limitées

Problématique

Résolution exacte

Algorithme de liste

Cas de ressources financières

Problématique

L'offre et la demande

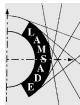
Algorithme de décalage

Contexte

Un projet consiste en un ensemble de n tâches liées par des contraintes de succession ou de précedence

Objectif

- Calculer la durée minimale du projet, les ressources étant supposées illimitées
⇒ Minimiser $(t_{n+1} - t_0)$ sous les contraintes de potentiels
- Déterminer les dates de début au plus tôt et au plus tard des tâches
- Déterminer les tâches critiques



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition

Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

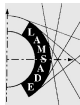
Problématique
L'offre et la demande
Algorithme de décalage

Formulation mathématique

Déterminer $(t_0, t_1, \dots, t_n, t_{n+1})$ de façon à
Minimiser $(t_{n+1} - t_0)$

s.c.

- 1** Contraintes de potentiel : $t_j - t_i \geq a_{ij}$
- 2** Contraintes de non négativité : $t_0, t_1, \dots, t_n, t_{n+1} \geq 0$



Plan

Problème central de l'ordonnement : ressources illimitées

Définition

Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Formulation mathématique

Déterminer $(t_0, t_1, \dots, t_n, t_{n+1})$ de façon à
Minimiser $(t_{n+1} - t_0)$

s.c.

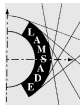
1 Contraintes de potentiel : $t_j - t_i \geq a_{ij}$

2 Contraintes de non négativité : $t_0, t_1, \dots, t_n, t_{n+1} \geq 0$

C'est un PL

MAIS, IL Y A PLUS SIMPLE !

⇒ **Méthode potentiels-tâches** et méthode PERT



Plan

Problème central de
l'ordonnement :
ressources illimitées

Définition

Modélisation

Résolution

Cas général :
ressources limitées

Problématique

Résolution exacte

Algorithme de liste

Cas de ressources
financières

Problématique

L'offre et la demande

Algorithme de décalage

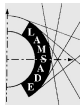
Modélisation avec un graphe potentiels-tâches

On utilise un graphe orienté $G = (X, U)$

- X : ensemble des sommets : tâches
- U : ensemble des arcs : contraintes de potentiels

$$U = \{(i, j) \in X \times X, \exists \text{ contrainte } t_j - t_i \geq a_{ij}\}$$

La valuation d'un arc (i, j) est $v_{ij} = a_{ij}$



Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

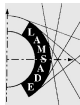
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Exemple illustratif

Rubrique	Tâche	Description	Durée	Tâches préc.
	Début	Lancement du projet	0	-
Sols	A	Dépose ancien carrelage	6	J
	B	Pose carrelage	4	A
	C	Joints carrelage	2	B
Murs	D	Décollage ancien papier	8	J
	E	Pose faïence	6	D
	F	Pose nouveau papier	4	E
Plomberie	G	Dépose ancien évier	1	Début
	H	déplacement arriv. et évac.	6	G
	I	Pose et raccordement Evier	2	M
Mobilier	J	Dépose ancien éléments	4	G
	K	Assemb. caissons et tiroirs	8	Début
	L	Pose éléments bas	6	B,J,H,K
	M	Pose plan de travail	4	L
	N	Etanchéité plan de travail	1	E,M
	O	Pose éléments hauts	1	E,J
	Fin	Fin du projet	0	C,F,I,N,O



Plan

Problème central de l'ordonnement : ressources illimitées

Définition

Modélisation

Résolution

Cas général : ressources limitées

Problématique

Résolution exacte

Algorithme de liste

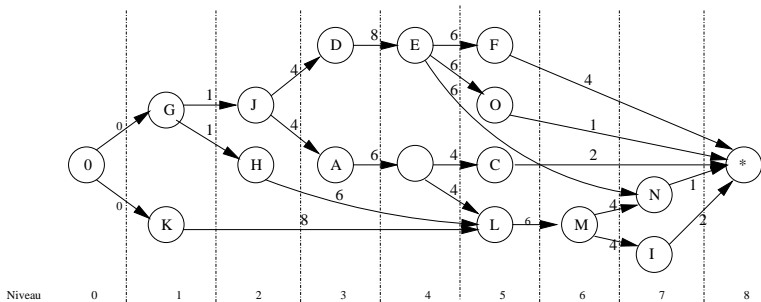
Cas de ressources financières

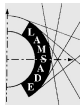
Problématique

L'offre et la demande

Algorithme de décalage

Graphe correspondant





Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

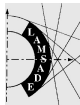
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Recherche d'un ordonnancement réalisable

Théorème

Il existe un ordonnancement réalisable ssi \nexists de circuit de valeur strictement positive dans G



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Recherche d'un ordonnancement réalisable

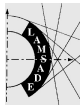
Théorème

Il existe un ordonnancement réalisable ssi \nexists de circuit de valeur strictement positive dans G

Si la condition d'existence est vérifiée alors, il existe en général plusieurs ordonnancements réalisables de durée minimale.

On distinguera deux cas particuliers :

- 1** Ordonnancement au plus tôt
- 2** Ordonnancement au plus tard (avec une date limite d'achèvement du projet imposée)



Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

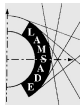
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Dates de début au plus tôt (1)

Définition

La date de début au plus tôt t_i d'une tâche i est égale à la **longueur du plus long chemin** de la tâche Début (ou 0) à i .



Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Dates de début au plus tôt (1)

Définition

La date de début au plus tôt \underline{t}_i d'une tâche i est égale à la longueur du plus long chemin de la tâche Début (ou 0) à i .

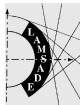
Théorème

Dans le cas d'un graphe sans circuit, on a

$$\underline{t}_0 = 0$$

$$\underline{t}_i = \max_{j \in \Gamma^{-1}(i)} (\underline{t}_j + v_{j,i})$$

$v_{j,i}$ est la valuation de l'arc (j, i) (par exemple la durée de j)
 $\Gamma^{-1}(i)$ est l'ensemble des prédécesseurs de i .



Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

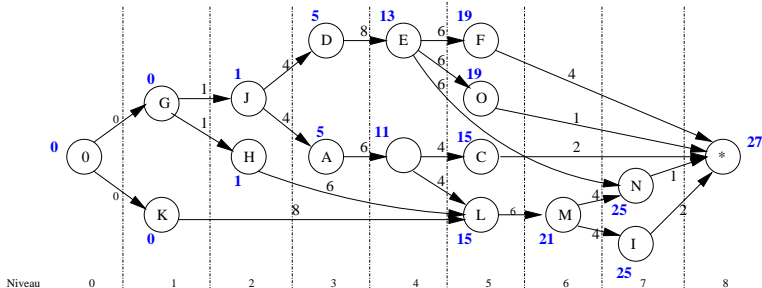
Cas général : ressources limitées

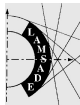
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Dates de début au plus tôt (2)





Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Dates de début au plus tard (1)

On souhaite terminer le projet au plus tard à une date

$$D \geq t_{n+1}$$

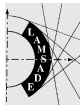
Définition

La date de début au plus tard \bar{t}_i de i est la date maximum à laquelle on peut exécuter i sans retarder le chantier (date D).

Théorème

$$\bar{t}_{n+1} = D$$

$$\bar{t}_i = \bar{t}_{n+1} - \text{valeur d'un plus long chemin entre } i \text{ et } n + 1$$



Plan

Problème central de
l'ordonnement :
ressources illimitées

Définition
Modélisation
Résolution

Cas général :
ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources
financières
Problématique
L'offre et la demande
Algorithme de décalage

Dates de début au plus tard (2)

Théorème

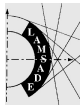
Dans le cas d'un graphe sans circuit, on a

$$\bar{t}_* = D$$

$$\bar{t}_i = \min_{j \in \Gamma(i)} \{\bar{t}_j - v_{i,j}\}$$

$v_{i,j}$ est la valuation de l'arc (i, j) (par exemple la durée de i)

$\Gamma(i)$ est l'ensemble des successeurs de i .



Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

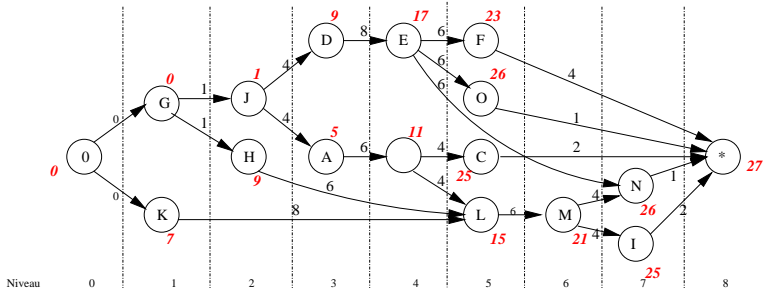
Cas général : ressources limitées

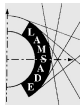
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Dates de début au plus tard (3)





Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Marges totales et chemin critique (1)

Définition

La **marge totale** d'une tâche i est le retard total qu'on peut se permettre sur i sans remettre en cause la date de fin du projet.

$$MT_i = \bar{t}_i - \underline{t}_i$$

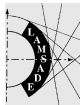
Définition

Les **tâches critiques** ont une marge nulle (par extension si

$$MT_i = MT_{n+1})$$

⇒ Tout retard sur leur exécution entraîne un retard global sur le projet.

Un **chemin est critique** s'il relie Début à Fin et s'il ne contient que des tâches critiques.



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

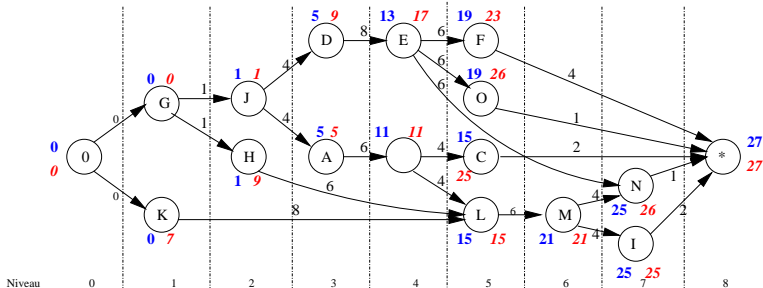
Cas général : ressources limitées

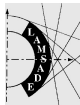
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Marges totales et chemin critique (2)





Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

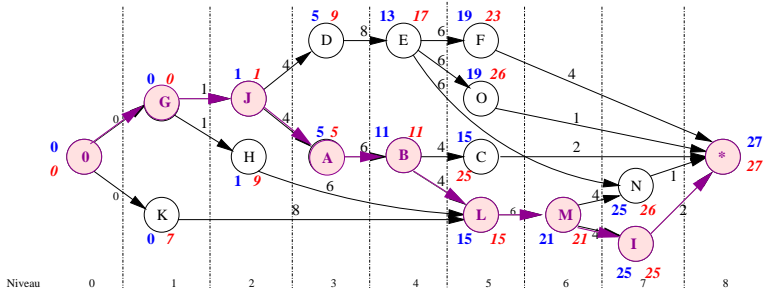
Cas général : ressources limitées

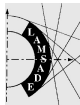
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Marges totales et chemin critique (3)





Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

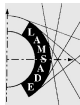
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Marges libres

Définition

La **marge libre** d'une tâche i est le retard total qu'on peut se permettre sur i sans retarder l'exécution d'une autre tâche (par rapport aux dates de début au plus tôt).



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Marges libres

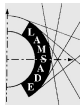
Définition

La **marge libre** d'une tâche i est le retard total qu'on peut se permettre sur i sans retarder l'exécution d'une autre tâche (par rapport aux dates de début au plus tôt).

Théorème

$$ML_i = \min_{j \in \Gamma(i)} \{ \underline{t}_j - \underline{t}_i - v_{i,j} \}$$

$v_{i,j}$ est la valuation de l'arc (i, j) (par exemple la durée de i)
 $\Gamma(i)$ est l'ensemble des successeurs de i .



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

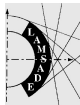
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Marges libres (1)

Définition

La **marge libre** d'une tâche i est le retard total qu'on peut se permettre sur i sans retarder l'exécution d'une autre tâche (par rapport aux dates de début au plus tôt).



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition

Modélisation

Résolution

Cas général : ressources limitées

Problématique

Résolution exacte

Algorithme de liste

Cas de ressources financières

Problématique

L'offre et la demande

Algorithme de décalage

Marges libres (1)

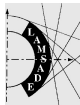
Définition

La **marge libre** d'une tâche i est le retard total qu'on peut se permettre sur i sans retarder l'exécution d'une autre tâche (par rapport aux dates de début au plus tôt).

Théorème

$$ML_i = \min_{j \in \Gamma(i)} \{ \underline{t}_j - \underline{t}_i - v_{i,j} \}$$

$v_{i,j}$ est la valuation de l'arc (i, j) (par exemple la durée de i)
 $\Gamma(i)$ est l'ensemble des successeurs de i .



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

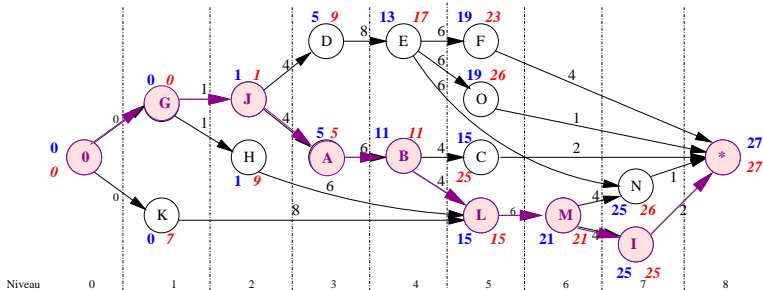
Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

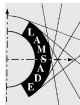
Problématique
L'offre et la demande
Algorithme de décalage

Problématique



La tâches D et A doivent être effectuées par la même personne.

⇒ On doit choisir d'effectuer D avant A ou bien A avant D



Plan

Problème central de
l'ordonnancement :
ressources illimitées

Définition
Modélisation
Résolution

Cas général :
ressources limitées

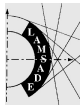
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources
financières

Problématique
L'offre et la demande
Algorithme de décalage

Schéma général

- 1 Enumérer les différentes possibilités :
 - Si A est avant D alors on rajoute un arc de A vers D valué par la durée de A ,
 - Si D est avant A alors on rajoute un arc de D vers A valué par la durée de D
- 2 Résoudre les problèmes correspondants
- 3 Choisir la meilleure décision



Plan

Problème central de
l'ordonnancement :
ressources illimitées

Définition
Modélisation
Résolution

Cas général :
ressources limitées

Problématique
Résolution exacte
Algorithme de liste

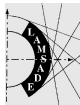
Cas de ressources
financières

Problématique
L'offre et la demande
Algorithme de décalage

Schéma général

- 1 Enumérer les différentes possibilités :
 - Si A est avant D alors on rajoute un arc de A vers D valué par la durée de A ,
 - Si D est avant A alors on rajoute un arc de D vers A valué par la durée de D
- 2 Résoudre les problèmes correspondants
- 3 Choisir la meilleure décision

Lorsque le nombre de tâches partageant les mêmes ressources est trop important le problème devient très difficile à résoudre car le nombre de possibilités devient énorme !



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

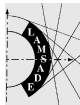
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Approche simple de résolution

Algorithme de liste

- Les conflits entre les tâches utilisant la même ressource sont résolus par des **règles de priorité statiques ou dynamiques**
- Exemples de règles de priorité
 - Plus petite date de début au plus tard d'abord
 - Plus petite date de fin au plus tard d'abord
 - Plus petite durée d'abord
- Les tâches sont ordonnancées en incrémentant le temps à partir de 0.



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

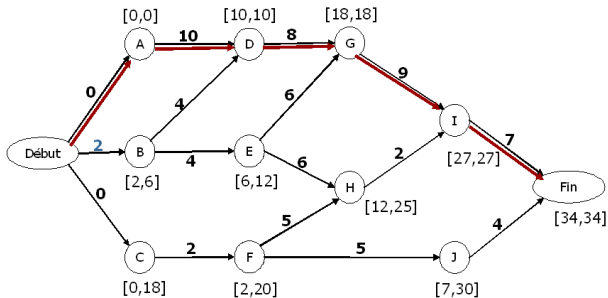
Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

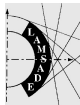
Problématique
L'offre et la demande
Algorithme de décalage

Autre exemple



	A	B	C	D	E	F	G	H	I	J	Disponibilit
Ressource R1	3	3	1	1	1	2	3	2	1	2	5
Ressource R2	0	0	0	1	1	1	0	1	0	0	1

Appliquer l'algorithme de liste en donnant la priorité à la tâche qui possède la plus petite date de début au plus tard.



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

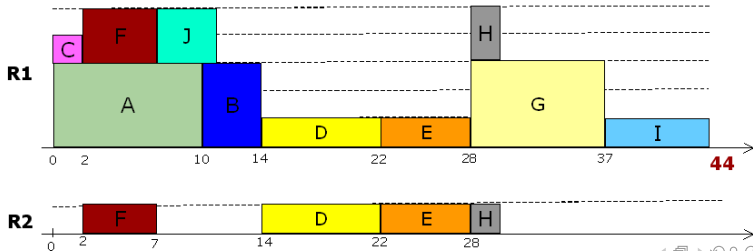
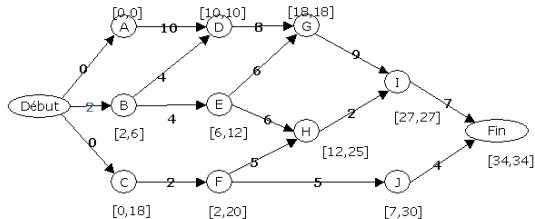
Cas général : ressources limitées

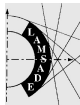
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Diagramme de Gantt





Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

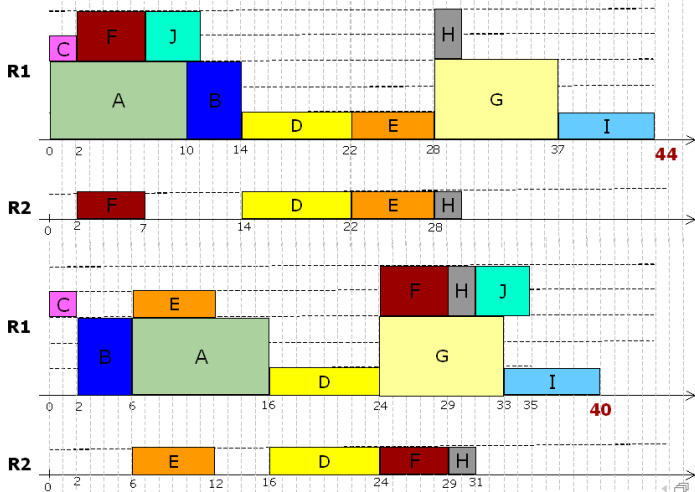
Cas général : ressources limitées

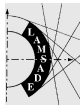
Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Solution trouvée vs solution optimale





Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Problématique

- Les tâches $\{1, 2, \dots, n\}$ ont besoin d'une seule ressource : l'**argent**
 - La ressource est alimentée à des dates u_1, \dots, u_q et selon des quantités b_1, \dots, b_q
 - Une tâche i requiert, à sa date de début d'exécution, une quantité a_i de la ressource
- Ordonnancement de durée minimale en respectant les
 - contraintes de ressource
 - contraintes de précédence entre tâches



Plan

Problème central de l'ordonnement : ressources illimitées

Définition
Modélisation
Résolution

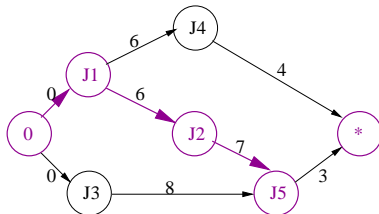
Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

Cas de ressources financières

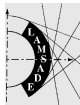
Problématique
L'offre et la demande
Algorithme de décalage

Exemple



- **Demande** : $\forall i \in \{J1, J2, J3, J4, J5\}, a_i = 5$
- **Offre**

u_i	b_i
0	15
10	5
16	10



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

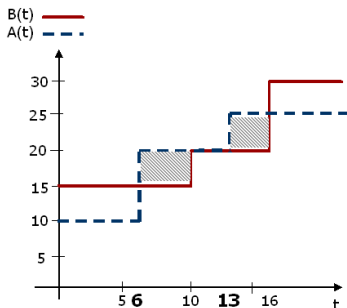
Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

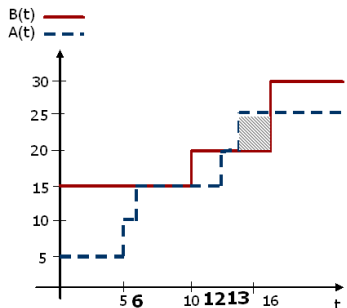
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

L'offre et la demande

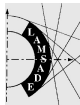


L'offre et la demande pour l'ordonnancement au plus tôt



L'offre et la demande pour l'ordonnancement au plus tard

Aucun des ordonnancements n'est admissible



Plan

Problème central de l'ordonnancement : ressources illimitées

Définition
Modélisation
Résolution

Cas général : ressources limitées

Problématique
Résolution exacte
Algorithme de liste

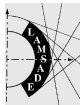
Cas de ressources financières

Problématique
L'offre et la demande
Algorithme de décalage

Algorithme de décalage

- On peut montrer que si un ordonnancement est admissible, tout ordonnancement au plus tard de même durée l'est aussi
- **Algorithme de décalage**
 - Calculer un ordonnancement au plus tard
 - S'il n'est pas admissible, calculer la quantité minimale δ^* dont il faut décaler l'ordonnancement au plus tard pour que l'ordonnancement résultant soit admissible

⇒ L'algorithme de décalage permet d'obtenir un ordonnancement optimal



Plan

Contexte

Classification

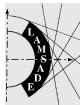
Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...
... des algorithmes
... des problèmes

Partie 3 : Ordonnancement d'ateliers – contexte et classification

- 9** Contexte
- 10** Classification des problèmes d'ordonnancement
 - Schémas de classification
 - Les environnements machines
 - Les tâches et leurs caractéristiques
 - Les critères d'optimisation
- 11** Les différentes classes d'ordonnancement
- 12** Quelques notions de la théorie de la complexité
 - Complexité des algorithmes
 - Complexité des problèmes



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

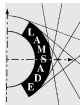
Les différentes classes d'ordonnancement

Complexité ...

... des algorithmes
... des problèmes

Contexte

- On a un ensemble \mathcal{J} de tâches ou de travaux (ou jobs) à exécuter
- Les ressources sont des machines et ne peuvent exécuter qu'une tâche à la fois
- Les critères font intervenir les dates de fin d'exécution, les dates de livraison, les stocks d'en cours ... et les ordres de fabrication peuvent avoir des poids (importance) différents



Plan

Contexte

Classification

Schémas de classification

Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

... des algorithmes

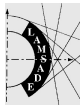
... des problèmes

Schémas de classification

Nous suivons les schémas de classification proposés par (Graham et al, 1979).

Classification à trois champs $\alpha|\beta|\gamma$

- α : environnement machine
- β : les caractéristiques des tâches
- γ : le (ou les) critère(s) à optimiser



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

... des algorithmes
... des problèmes

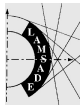
D'une façon générale ...

On doit exécuter n tâches ou n travaux (*jobs*).

Le champ α est décomposé en deux sous-champs α_1 et α_2 .

Selon les valeurs prises par α_1 , on distingue :

- Les problèmes à une machine
- Les problèmes à machines parallèles
- Les problèmes d'ateliers
 - Ateliers à cheminement unique (*flowshop*)
 - Ateliers à cheminements multiples (*jobshop*)
 - Ateliers à cheminements libres (*openshop*)
 - ...
- L'ordonnancement de projet sous contraintes de ressources



Plan

Contexte

Classification

Schémas de classification

Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

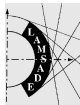
... des algorithmes

... des problèmes

Problèmes à une machine

Toute tâche $J_j, j \in \{1, \dots, n\}$ de durée p_j (*processing time*) s'exécute sur une machine qui ne peut traiter plus qu'une tâche à la fois.

Le champ α_1 est absent et $\alpha_2 = 1$.



Plan

Contexte

Classification

Schémas de
classificationLes environnements
machinesCaractéristiques de
tâchesLes critères
d'optimisation

Les différentes

classes

d'ordonnement

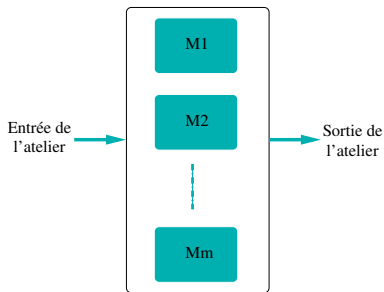
Complexité ...

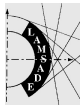
... des algorithmes

... des problèmes

Problèmes à machines parallèles (1)

Toute tâche $J_j, j \in \{1, \dots, n\}$ peut être exécutée indifféremment sur une des m machines mises en parallèle.





Plan

Contexte

Classification

Schémas de classification

Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes

d'ordonnancement

Complexité ...

... des algorithmes

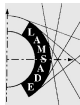
... des problèmes

Problèmes à machines parallèles (2)

$p_{i,j}$ est la durée d'exécution de J_j sur la machine

$M_i, i = 1, \dots, m.$

- si $\alpha_1 = P$ alors **machines identiques** $\Rightarrow \forall i, p_{i,j} = p_j$
- si $\alpha_1 = Q$ alors **machines uniformes** $\Rightarrow \forall i, p_{i,j} = p_j/s_i$ où s_i est la vitesse de traitement M_i
- si $\alpha_1 = R$ alors **machines indépendantes** $\Rightarrow \forall i, p_{i,j} = p_j/s_{i,j}$ où $s_{i,j}$ est la vitesse de traitement de la tâche J_j par la machine M_i
- si α_2 est un entier positif, le nombre de machines est supposé constant. Si α_2 est absent alors ce nombre est supposé arbitraire.



Plan

Contexte

Classification

Schémas de classification

Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes

d'ordonnement

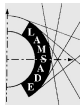
Complexité ...

... des algorithmes

... des problèmes

Problèmes d'ateliers

- m machines différentes $M_i, i \in \{1, \dots, m\}$
- n travaux (jobs) $J_j, j \in \{1, \dots, n\}$.
 - Chaque job J_j est décrit par n_j tâches ou **opérations** $O_{i,j}, i \in \{1, \dots, n_j\}$
 - La durée d'une opération $O_{i,j}$ est $p_{i,j}$
 - La machine qui exécute l'opération $O_{i,j}$ du job est notée $M(O_{i,j})$ ou $M_{i,j}$.
 - **Les opérations d'un même job ne peuvent pas être exécutées simultanément**
- α_2 (voir machines parallèles)



Plan

Contexte

Classification

Schémas de classification

Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

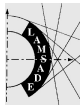
... des algorithmes

... des problèmes

Ateliers à cheminement unique : Flowshop



- Chaque Job est constitué de m opérations et l'ordre de passage sur les différentes machines est le même pour tous les jobs $J_j : O_{1,j} \rightarrow O_{2,j} \rightarrow, \dots, \rightarrow O_{m,j}$ et $M_{i,j} = M_i$
- $\alpha_1 = F$



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnement

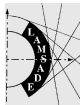
Complexité ...

... des algorithmes
... des problèmes

Ateliers à cheminements quelconques : Jobshop

- Le nombre d'opérations n'est pas forcément le même pour tous les jobs
- Chaque job a son propre ordre de passage sur les machines
- $\alpha_1 = J$
- Exemple

J_j	J_1			J_2			J_3	
$O_{i,j}$	$O_{1,1}$	$O_{2,1}$	$O_{3,1}$	$O_{1,2}$	$O_{2,2}$	$O_{3,2}$	$O_{1,3}$	$O_{2,3}$
$M_{i,j}$	M_1	M_2	M_3	M_2	M_1	M_3	M_3	M_2
$p_{i,j}$	3	2	5	4	2	2	2	3



Plan

Contexte

Classification

Schémas de classification

Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes d'ordonnancement

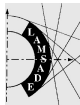
Complexité ...

... des algorithmes

... des problèmes

Ateliers à cheminements libres : Openshop

- Le nombre d'opérations n'est pas forcément le même pour tous les jobs
- L'ordre de passage sur les machines est totalement libre
- $\alpha_1 = O$



Plan

Contexte

Classification

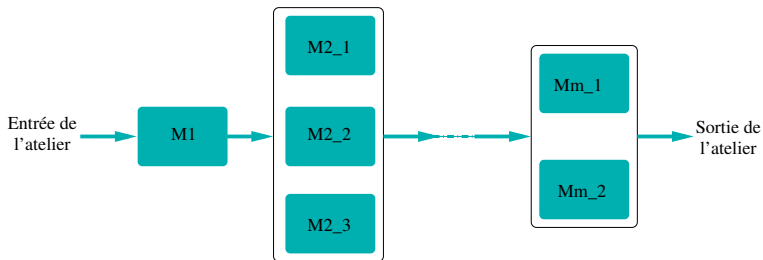
Schémas de
classification**Les environnements
machines**Caractéristiques de
tâchesLes critères
d'optimisationLes différentes
classes
d'ordonnancement

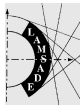
Complexité ...

... des algorithmes

... des problèmes

Autres ateliers : Flowshop hybride





Plan

Contexte

Classification

Schémas de classification
Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes d'ordonnancement

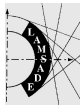
Complexité ...

... des algorithmes
... des problèmes

Les tâches et leurs caractéristiques

Chaque tâche J_j ou chaque job J_j peut être caractérisé par

- une date de début au plus tôt r_j (**release date**)
- une durée p_j , pour la tâche, ou $p_{i,j}$, pour l'opération i du job (**processing time**)
- une date de fin souhaitée d_j (**due date**)
- une date de fin obligatoire \tilde{d}_j (**deadline**)
- un poids relatif w_j (**weight**) \Rightarrow importance ou poids
- Il peut y avoir des contraintes de précédence entre les tâches. Ces contraintes sont représentées par un graphe $G = (\mathcal{J}, A)$.



Plan

Contexte

Classification

Schémas de classification
Les environnements machines

Caractéristiques de tâches

Les critères d'optimisation

Les différentes classes d'ordonnement

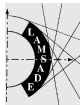
Complexité ...

... des algorithmes
... des problèmes

Les tâches et leurs caractéristiques

$\beta = \beta_1\beta_2\beta_3\beta_4 \dots$ dans $\alpha|\beta|\gamma$

- $\beta_1 = pmtn$ si la **préemption** des tâches est autorisée, sinon β_1 est absent
- S'il y a des **contraintes de précédence** entre les tâches $\beta_2 \in \{prec, chain, in - tree, out - tree\}$, sinon β_2 est vide
- $\beta_3 = r_j$ si les **dates de début au plus tôt** r_j (ou dates de disponibilité) des tâches ne sont pas forcément identiques, sinon $(\forall j, r_j = 0)$ β_3 est absent
- $\beta_4 = \tilde{d}_j$ si la tâche ou le job possède une **date de fin obligatoire**



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnement

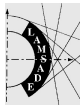
Complexité ...

... des algorithmes
... des problèmes

Quelques mesures

Les critères d'optimisation s'expriment en fonction des dates de fin des tâches (ou jobs) C_j (**completion times**). Les critères sont généralement exprimés en fonction des mesures suivantes

- le retard algébrique $L_j = C_j - d_j$ (**lateness**)
- le retard absolu $T_j = \max(0, C_j - d_j)$ (**tardiness**)
- l'avance $E_j = \max(0, d_j - C_j)$ (**earliness**)
- le pénalité unitaire de retard $U_j = 0$ si $C_j \leq d_j$, $U_j = 1$ sinon
- le durée de séjour dans l'atelier $F_j = C_j - r_j$
- une fonction générique $f_j(t)$ donnant le coût induit si on termine J_j à t



Plan

Contexte

Classification

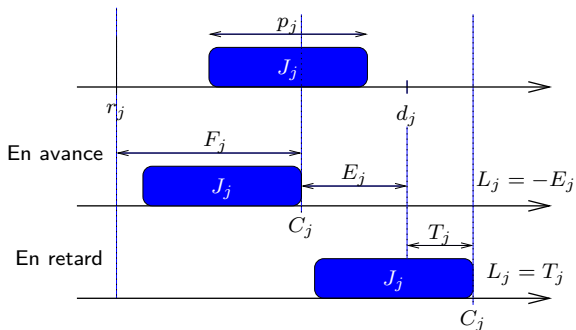
- Schémas de classification
- Les environnements machines
- Caractéristiques de tâches
- Les critères d'optimisation

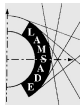
- Les différentes classes d'ordonnancement

Complexité ...

- ... des algorithmes
- ... des problèmes

Illustration





Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

... des algorithmes
... des problèmes

Les critères d'optimisation

On cherche alors à minimiser un ou plusieurs critères

$F(C_1, \dots, C_n)$

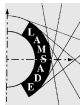
- une fonction générique de **coût maximum**

$$f_{\max} = \max_j \{f_j(C_j)\}$$

- la durée totale de l'ordonnancement $C_{\max} = \max_j \{C_j\}$
- le retard algébrique maximum $L_{\max} = \max_j L_j$
- le retard maximum $T_{\max} = \max_j T_j$

- une fonction générique de **coût total** $f_{\Sigma} = \sum_j \{f_j(C_j)\}$

- la somme (pondérée) des dates de fin $\sum_j (w_j)C_j$
- la somme (pondérée) des retards $\sum_j (w_j)T_j$
- le nombre (pondéré) des tâches (ou jobs) en retard $\sum_j (w_j)U_j$
- la durée moyenne de séjour $\sum_j F_j$
- la somme (pondérée) des avances et des retards $\sum_j \alpha_j E_j + \beta_j T_j$



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

... des algorithmes
... des problèmes

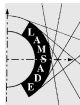
Critères réguliers

Définition

Un **critère** $F(C_1, \dots, C_n)$ est dit **régulier** si et seulement si F est une fonction croissante des dates de fin des tâches (ou jobs)

Corollaire

- Les critères C_{\max} , L_{\max} , T_{\max} , $\sum_j (w_j)C_j$, $\sum_j (w_j)T_j$, $\sum_j (w_j)U_j$, $\sum_j (w_j)F_j$ sont réguliers
- Le critère $\sum_j \alpha_j E_j + \beta_j T_j$ n'est pas régulier



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnement

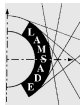
Complexité ...

... des algorithmes
... des problèmes

Les différentes classes d'ordonnement

Quatre catégories ou classes (par ordre d'inclusion)

- les ordonnancements quelconques
- les ordonnancements **semi-actifs**
- les ordonnancements **actifs**
- les ordonnancements **sans délai**



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnement

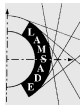
Complexité ...

... des algorithmes
... des problèmes

Les différentes classes d'ordonnement

Définitions

- Un ordonnancement est dit **semi-actif** si aucune tâche ne peut être exécutée plus tôt sans changer l'ordre d'exécution sur les ressources ou violer de contrainte (de précedence, date de début au plus tôt, ...)
- Un ordonnancement est dit **actif** si aucune tâche ne peut être exécutée plus tôt sans retarder une autre tâche ou violer de contrainte
- Un ordonnancement **sans délai** est un ordonnancement pour lequel aucune machine n'est laissée inactive alors qu'elle pourrait commencer une tâche disponible



Plan

Contexte

Classification

- Schémas de classification
- Les environnements machines
- Caractéristiques de tâches
- Les critères d'optimisation

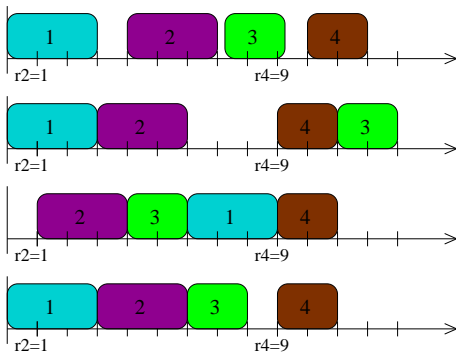
Les différentes classes d'ordonnement

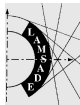
Complexité ...

- ... des algorithmes
- ... des problèmes

Les différentes classes d'ordonnement

Exemple : 4 tâches, à exécuter sur une machine, telles que $r_1 = r_3 = 0$, $r_2 = 1$, $r_4 = 9$; $p_1 = p_2 = 3$, $p_3 = 2$, $p_4 = 2$





Plan

Contexte

Classification

- Schémas de classification
- Les environnements machines
- Caractéristiques de tâches
- Les critères d'optimisation

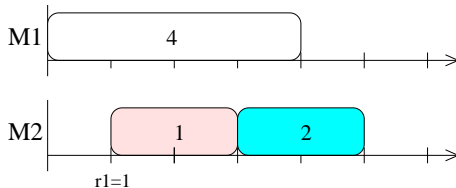
Les différentes classes d'ordonnement

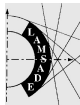
Complexité ...

- ... des algorithmes
- ... des problèmes

Les différentes classes d'ordonnement

Exemple : 3 tâches, à exécuter sur deux machines en parallèle, telles que $r_2 = r_3 = 0$, $r_1 = 1$; $p_1 = p_2 = 2$, $p_3 = 4$





Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnancement

Complexité ...

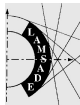
... des algorithmes
... des problèmes

Complexité des algorithmes

- Un même problème peut généralement être résolu par plusieurs algorithmes
⇒ il faut comparer entre ces algorithmes
- La comparaison se base sur le **temps de calcul** et sur l'**espace mémoire** requis par l'algorithme

Définition

On appelle **complexité en temps d'un algorithme dans le pire cas**, la fonction $f(n)$ qui donne une borne supérieure du nombre d'opérations élémentaires effectuées par l'algorithme lorsque la taille de l'entrée est n .



Plan

Contexte

Classification

- Schémas de classification
- Les environnements machines
- Caractéristiques de tâches
- Les critères d'optimisation

- Les différentes classes d'ordonnement

Complexité ...

- ... des algorithmes
- ... des problèmes

Complexité des algorithmes

Définition

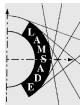
On dit que $f(n) \in \mathcal{O}(g(n))$, s'il existe une constante $c > 0$ et un entier n_0 tels que $\forall n \geq n_0, |f(n)| \leq c|g(n)|$

Définition

Un algorithme est de **complexité polynomiale** lorsque $f(n) \in \mathcal{O}(p(n))$ et p est un polynôme en n , c'est-à-dire, il existe une constante k telle que $f(n) \in \mathcal{O}(n^k)$

Définition

Un algorithme dont la fonction complexité $f(n)$ ne peut pas être majorée par un polynôme en n est dit **exponentiel**



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

Les différentes classes d'ordonnement

Complexité ...

... des algorithmes
... des problèmes

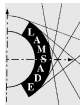
Complexité des problèmes

Distinction entre problème de décision et problème d'optimisation

- Un **problème de décision** est un problème pour lequel une solution est soit "oui" soit "non"
- Un **problème d'optimisation** est un problème pour lequel on doit chercher à déterminer une solution qui optimise un critère
- A chaque problème d'optimisation on peut associer un problème de décision

Définition

Un problème de décision appartient à la classe \mathcal{P} , s'il peut être résolu par un algorithme polynomial en n



Plan

Contexte

Classification

- Schémas de classification
- Les environnements machines
- Caractéristiques de tâches
- Les critères d'optimisation

- Les différentes classes d'ordonnement

Complexité ...

- ... des algorithmes
- ... des problèmes

Complexité des problèmes

Définition

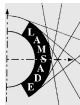
Un problème de décision appartient à la **classe \mathcal{NP}** s'il peut être résolu par un **algorithme non déterministe polynomial**.

Définition

On considère deux problèmes de décisions Q et R . On dit que **Q se réduit polynomialement à R** , et on note $Q \propto R$, s'il existe une fonction polynomiale g qui transforme toute instance de Q en une instance de R telle que x est une réponse "oui" de Q si et seulement si $g(x)$ est une réponse "oui" de R .

Définition

Un problème R est **\mathcal{NP} -complet** ssi $R \in \mathcal{NP}$ et $\forall Q \in \mathcal{NP}, \exists \propto$ telle que $Q \propto R$.



Plan

Contexte

Classification

Schémas de classification
Les environnements machines
Caractéristiques de tâches
Les critères d'optimisation

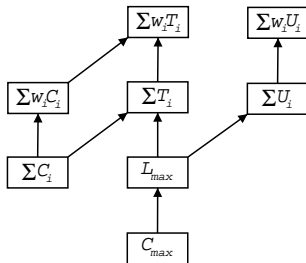
Les différentes classes d'ordonnement

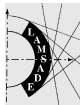
Complexité ...

... des algorithmes
... des problèmes

Complexité des problèmes

- Un problème d'optimisation est dit **\mathcal{NP} -difficile** si le problème de décision associé est \mathcal{NP} -complet
- Il existe un certain nombre de résultats dans la littérature qui montrent les liens, sous forme d'**arbres de réduction**, entre différents problèmes d'ordonnement.
- Arbre de réduction en fonction des **critères d'optimisation**





Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

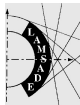
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Résolution des problèmes d'ordonnement

- 13** Algorithmes à base de règles de priorité
 - Problèmes à une machine
 - Problèmes de type flowshop
 - Problèmes à machines parallèles
 - Limitation des règles de priorité
- 14** Algorithmes plus "élaborés"
- 15** Approches pour résoudre les problèmes NP-difficiles
 - Méthodes constructives
 - Heuristiques par voisinage
 - Procédures par évaluation et séparation
 - Résolution du problème $1|r_j|L_{\max}$
 - Problème $J||C_{\max}$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Ensembles dominants

Definition

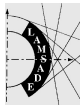
Un **ensemble** d'ordonnements est dit **dominant** pour un problème d'ordonnement donné si et seulement si quelques soient les données du problème, une solution optimale est contenue dans cet ensemble

Theorem

*L'ensemble des **ordonnements actifs** est dominant pour les **critères réguliers***

Corollaire

*L'ensemble des **ordonnements semi-actifs** est dominant pour les **critères réguliers***



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

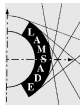
Ensembles dominants

Corollaire

Un ordonnancement est complètement déterminé par la séquence des tâches au niveau de chaque machine

Attention

Un ensemble de séquences (au niveau des machines) ne représente pas forcément un ordonnancement admissible



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation

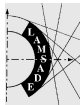
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

- Toutes les tâches sont disponibles à l'instant $t = 0$
($\forall j, r_j = 0$)
- w_j est le poids de la tâche J_j (ou tout simplement j)
- C_j la date de fin de j (c'est une inconnue)
- **Un ordonnancement est complètement déterminé par une séquence des tâches**
- Minimiser $\sum w_j C_j$ revient à minimiser les encours totaux (pondérés)



Plan

Algorithmes à base
de règles de priorité

**Problèmes à une
machine**

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

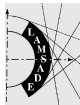
$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2

Exemple de séquence : $\pi_1 = (1, 2, 3, 4, 5)$



Plan

Algorithmes à base
de règles de priorité

**Problèmes à une
machine**

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes
NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

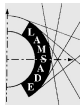
Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2

Exemple de séquence : $\pi_1 = (1, 2, 3, 4, 5)$

$$\sum w_j C_j(\pi_1) = 1 * 2 + 3 * 5 + 2 * 11 + 1 * 16 + 2 * 17 = 89$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

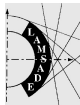
Problème 1 || $\sum w_j C_j$

Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2

Exemple de séquence : $\pi_1 = (1, 2, 3, 4, 5)$

$$\sum w_j C_j(\pi_1) = 1 * 2 + 3 * 5 + 2 * 11 + 1 * 16 + 2 * 17 = 89$$

Question : Peut-on faire mieux ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

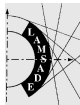
Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2

Exemple de séquence : $\pi_1 = (1, 2, 3, 4, 5)$

$$\sum w_j C_j(\pi_1) = 1 * 2 + 3 * 5 + 2 * 11 + 1 * 16 + 2 * 17 = 89$$

Question : Peut-on faire mieux ?

Réponse : oui !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

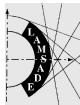
Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème 1|| $\sum w_j C_j$

Theorem

Règle de Smith : Ordonnancer les tâches dans l'ordre croissant des p_j/w_j minimise le stock d'encours (i.e. $\sum w_j C_j$)



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

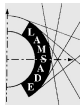
Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

Theorem

Règle de Smith : Ordonnancer les tâches dans l'ordre croissant des p_j/w_j minimise le stock d'encours (i.e. $\sum w_j C_j$)

Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2
p_j/w_j	2	1	3	5	0.5



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

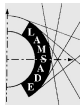
Theorem

Règle de Smith : Ordonner les tâches dans l'ordre croissant des p_j/w_j minimise le stock d'encours (i.e. $\sum w_j C_j$)

Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2
p_j/w_j	2	1	3	5	0.5

La séquence optimale pour le problème avec poids est

$$\pi_w^* = (5, 2, 1, 3, 4)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème 1 || $\sum w_j C_j$

Theorem

Règle de Smith : Ordonner les tâches dans l'ordre croissant des p_j/w_j minimise le stock d'encours (i.e. $\sum w_j C_j$)

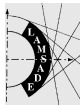
Tâche j	1	2	3	4	5
p_j	2	3	6	5	1
w_j	1	3	2	1	2
p_j/w_j	2	1	3	5	0.5

La séquence optimale pour le problème avec poids est

$$\pi_w^* = (5, 2, 1, 3, 4)$$

La séquence optimale pour le problème sans poids est

$$\pi_1^* = (5, 1, 2, 4, 3)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problèmes $1|r_j| \sum w_j C_j$ et $1|r_j, pmtn| \sum w_j C_j$

On associe une date de disponibilité r_j à une tâche j

Theorem

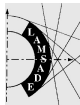
Le problème $1|r_j| \sum w_j C_j$ est NP-difficile

Theorem

Le problème $1|r_j, pmtn| \sum w_j C_j$ est NP-difficile

Theorem

Le problème $1|r_j, pmtn| \sum C_j$ peut être résolu en temps polynomial :



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problèmes $1|r_j|\sum w_j C_j$ et $1|r_j, pmtn|\sum w_j C_j$

On associe une date de disponibilité r_j à une tâche j

Theorem

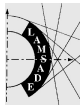
Le problème $1|r_j|\sum w_j C_j$ est NP-difficile

Theorem

Le problème $1|r_j, pmtn|\sum w_j C_j$ est NP-difficile

Theorem

Le problème $1|r_j, pmtn|\sum C_j$ peut être résolu en temps polynomial : Affecter la machine à la tâche de plus courte durée sous la condition suivante : quand une tâche arrive, elle préempte la machine si sa durée est plus petite que la durée résiduelle de la tâche en cours.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

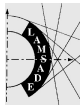
Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problèmes $1||L_{\max}$ et $1|r_j, pmtn|L_{\max}$

Règle

La règle de Jackson consiste à donner la priorité la plus élevée à la tâche disponible de plus petite date échu.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problèmes $1||L_{\max}$ et $1|r_j, pmtn|L_{\max}$

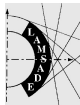
Règle

La *règle de Jackson* consiste à donner la priorité la plus élevée à la tâche disponible de plus petite date échu.

Definition

On désignera par **ordonnancement de Jackson** l'ordonnancement de liste obtenu en appliquant la règle de Jackson

Si $\forall j, r_j = 0$, on parle également d'ordre EDD : Earliest Due Date.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problèmes $1||L_{\max}$ et $1|r_j, pmtn|L_{\max}$

EXEMPLE

j	1	2	3	4	5	6	7
r_j	0	9	13	11	20	30	30
p_j	6	7	6	7	4	3	5
d_j	31	41	22	24	27	40	48

Ordonnancement de Jackson préemptif

Ordonnancement de Jackson non-préemptif



Plan

Algorithmes à base de règles de priorité

Problèmes à une machine

Problèmes de type flowshop

Problèmes à machines parallèles

Limitation des règles de priorité

Algorithmes plus "élaborés"

Approches pour résoudre les problèmes NP-difficiles

Méthodes constructives

Heuristiques par voisinage

Procédures par évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

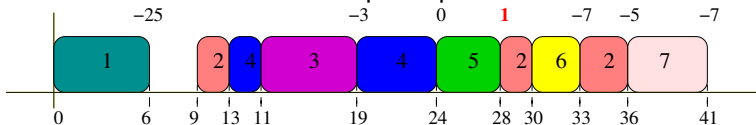
Problème $J||C_{\max}$

Problèmes $1||L_{\max}$ et $1|r_j, pmtn|L_{\max}$

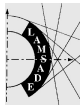
EXEMPLE

j	1	2	3	4	5	6	7
r_j	0	9	13	11	20	30	30
p_j	6	7	6	7	4	3	5
d_j	31	41	22	24	27	40	48

Ordonnancement de Jackson préemptif



Ordonnancement de Jackson non-préemptif



Plan

Algorithmes à base de règles de priorité

Problèmes à une machine

Problèmes de type flowshop

Problèmes à machines parallèles

Limitation des règles de priorité

Algorithmes plus "élaborés"

Approches pour résoudre les problèmes NP-difficiles

Méthodes constructives

Heuristiques par voisinage

Procédures par évaluation et séparation

Résolution du problème $1|r_j|L_{\max}$

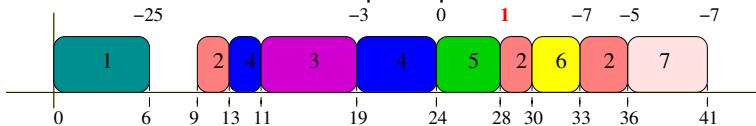
Problème $J||C_{\max}$

Problèmes $1||L_{\max}$ et $1|r_j, pmtn|L_{\max}$

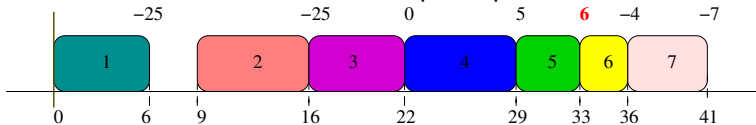
EXEMPLE

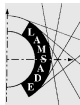
j	1	2	3	4	5	6	7
r_j	0	9	13	11	20	30	30
p_j	6	7	6	7	4	3	5
d_j	31	41	22	24	27	40	48

Ordonnancement de Jackson préemptif



Ordonnancement de Jackson non-préemptif





Plan

Algorithmes à base
de règles de priorité

**Problèmes à une
machine**

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

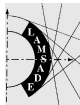
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problèmes $1||L_{\max}$ et $1|r_j, pmtn|L_{\max}$

- Montrer que la règle EDD est optimale pour $1||L_{\max}$
- Montrer que l'ordonnancement de Jackson est optimal pour $1|r_j, p_j = 1, pmtn|L_{\max}$
- Montrer que l'ordonnancement de Jackson est optimal pour $1|r_j, pmtn|L_{\max}$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $1|r_j|L_{\max}$

Theorem

Le problème $1|r_j|L_{\max}$ est NP-difficile au sens fort

Réduction à partir du problème 3-PARTITION qui est
NP-complet au sens fort

Definition

Etant donnés $n + 1 = 3m + 1$ entiers positifs a_1, \dots, a_{3m} et A
tels que $A/4 < a_j < A/2$, $j = 1, \dots, 3m$, et $\sum_{j=1}^{3m} a_j = mA$,
existe-t-il une partition de l'ensemble $\{1, \dots, 3m\}$ en m
sous-ensembles X_1, \dots, X_m , tel que $\sum_{j \in X_l} a_j = A$,
 $l = 1, \dots, m$?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

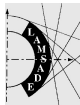
Problème $J||C_{\max}$

Problème $1|r_j|L_{\max}$

A toute instance du problème 3-PARTITION, on peut associer une instance du problème $1|r_j|L_{\max}$ à $n + m - 1$ tâches telles que

- $r_j = 0$, $p_j = a_j$ et $d_j = \sum_{j=1}^n a_j + m - 1$, pour $j \in \{1, \dots, n\}$
- $r_{n+k} = kA + k - 1$, $p_{n+k} = 1$ et $d_{n+k} = kA + k$ pour $k \in \{1, \dots, m - 1\}$

On montre que 3-PARTITION a une solution **si, et seulement si**, la valeur de la solution optimale, de l'instance construite du problème $1|r_j|L_{\max}$, est égale à 0.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes
NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

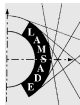
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $F2||C_{\max}$

- n tâches $J_j, j = 1, \dots, n$, à exécuter sur deux machines M_1 et M_2
- Cheminement unique : M_1 puis M_2
- a_j et b_j sont les durées de la tâche $J_j, j = 1, \dots, n$ sur les machines M_1 et M_2 , respectivement.
- Intuitivement
 - avoir des tâches exécutées le plus rapidement possible sur M_1 pour pouvoir les exécuter sur M_2 (M_2 reste inactive le minimum de temps possible)
 \implies Règle SPT sur M_1
 - Sur M_2 , on essaie d'exécuter les tâches dans l'ordre LPT pour que M_2 n'attende pas trop que M_1 lui fournisse des tâches



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

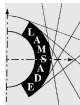
Problème $J||C_{\max}$

Problème $F2||C_{\max}$

Règle

Règle de Johnson

- Partitionner l'ensemble des tâches en 2 sous-ensembles
 - $A = \{j \in \mathcal{J}, a_j \leq b_j\}$
 - $B = \{j \in \mathcal{J}, a_j > b_j\}$
- Séquencer les tâches de A dans l'ordre croissant des a_j
 \Rightarrow séquence π_A
- Séquencer les tâches de B dans l'ordre décroissant des b_j
 \Rightarrow séquence π_B
- Fusionner les deux séquences $\Rightarrow \pi = \{\pi_A, \pi_B\}$
- On utilise le même ordre de passage sur les 2 machines



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

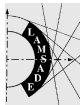
Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème $F2||C_{\max}$

Definition

Un ordonnancement pour lequel toutes les tâches sont exécutées dans un même ordre (sur les machines) est appelé **ordonnancement de permutation**



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

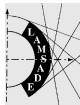
Problème $F2||C_{\max}$

Definition

Un ordonnancement pour lequel toutes les tâches sont exécutées dans un même ordre (sur les machines) est appelé **ordonnancement de permutation**

Lemme

Une instance du problème $F2||C_{\max}$ a toujours une solution optimale qui est un ordonnancement de permutation



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $F2||C_{\max}$

Definition

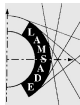
Un ordonnancement pour lequel toutes les tâches sont exécutées dans un même ordre (sur les machines) est appelé **ordonnancement de permutation**

Lemme

Une instance du problème $F2||C_{\max}$ a toujours une solution optimale qui est un ordonnancement de permutation

Theorem

La règle de Johnson est optimale pour $F2||C_{\max}$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

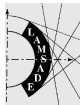
Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème $F2||C_{\max}$

Exemple

j	M_1	M_2
1	6	8
2	4	5
3	4	1
4	8	4



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $F2||C_{\max}$

Exemple

j	M_1	M_2
1	6	8
2	4	5
3	4	1
4	8	4

$$A = \{1, 2\} \text{ et } B = \{3, 4\}$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $F2||C_{\max}$

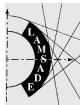
Exemple

j	M_1	M_2
1	6	8
2	4	5
3	4	1
4	8	4

$A = \{1, 2\}$ et $B = \{3, 4\}$

Séquence optimale $\pi^* = (2, 1, 4, 3)$

Dessiner le digramme de Gantt de l'ordonnement optimal



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les

problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

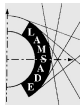
Problèmes à machines parallèles

Deux types de décision à prendre

1 affectation des tâches aux machines

2 séquençement de chaque machine

- d'une façon indépendante s'il n'y a pas de contraintes de précedence entre les tâches
- d'une "façon dépendante" sinon



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

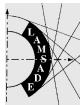
Problème $J||C_{\max}$

Problème $P||C_{\max}$

Theorem

Le problème $P||C_{\max}$ est NP-difficile

- On note C_{\max}^* la valeur de la solution optimale
- Montrer que
 - $C_{\max}^* \geq (1/m) * \sum_j p_j$
 - $C_{\max}^* \geq p_j, \forall j = 1, \dots, n$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

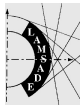
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $P|pmtn|C_{\max}$

- Montrer que la durée d'un ordonnancement optimal est supérieure ou égale à $B = \max(\max_j p_j, (1/m) * \sum_j p_j)$
- Proposer un algorithme qui construit un ordonnancement dont la durée est exactement B
- Calculer le nombre d'opérations nécessaires
- Conclure quant la complexité de l'algorithme



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

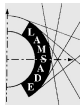
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $P|pmtn|C_{\max}$

- Montrer que la durée d'un ordonnancement optimal est supérieure ou égale à $B = \max(\max_j p_j, (1/m) * \sum_j p_j)$
- Proposer un algorithme qui construit un ordonnancement dont la durée est exactement B
- Calculer le nombre d'opérations nécessaires
- Conclure quant la complexité de l'algorithme



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

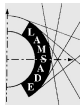
Procédures par
évaluation et séparation
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $P|pmtn|C_{\max}$

- Montrer que la durée d'un ordonnancement optimal est supérieure ou égale à $B = \max(\max_j p_j, (1/m) * \sum_j p_j)$
- Proposer un algorithme qui construit un ordonnancement dont la durée est exactement B
- Calculer le nombre d'opérations nécessaires
- Conclure quant la complexité de l'algorithme



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

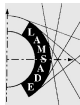
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $P|pmtn|C_{\max}$

- Montrer que la durée d'un ordonnancement optimal est supérieure ou égale à $B = \max(\max_j p_j, (1/m) * \sum_j p_j)$
- Proposer un algorithme qui construit un ordonnancement dont la durée est exactement B
- Calculer le nombre d'opérations nécessaires
- Conclure quant la complexité de l'algorithme



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

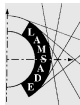
Problème $P||C_{\max}$

Theorem

Un algorithme de liste est un "2-approximation algorithm" pour $P||C_{\max}$

Theorem

LPT est un "4/3-approximation algorithm" pour $P||C_{\max}$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

**Limitation des règles de
priorité**

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

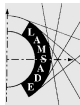
Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$

Problème $J||C_{\max}$

Voir exemple du livre



Plan

Algorithmes à base
de règles de priorité

- Problèmes à une machine
- Problèmes de type flowshop
- Problèmes à machines parallèles
- Limitation des règles de priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

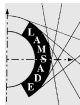
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $1||f_{\max}$

- On note $P = p_1 + p_2 + \dots + p_n$
- J_k la tâche telle que $f_k(P) = \min\{f_j(P), j = 1, \dots, n\}$
- Montrons qu'il \exists un ordonnancement optimal tel que J_k soit la dernière tâche exécutée
- $f_{\max}^*(S)$ la valeur optimale de la fonction objectif si on n'ordonne que les tâches de $S \subseteq \mathcal{J}$
- $f_{\max}^*(\mathcal{J}) \geq \min\{f_j(P), j = 1, \dots, n\}$
- $f_{\max}^*(\mathcal{J}) \geq f_{\max}^*(\mathcal{J} - \{J_j\})$
- La procédure "Least-Cost-Last" est optimale pour $1||f_{\max}$
- Le problème $1|prec|f_{\max}$ est-t-il polynomial?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

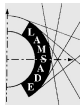
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $1||f_{\max}$

- On note $P = p_1 + p_2 + \dots + p_n$
- J_k la tâche telle que $f_k(P) = \min\{f_j(P), j = 1, \dots, n\}$
- Montrons qu'il \exists un ordonnancement optimal tel que J_k soit la dernière tâche exécutée
- $f_{\max}^*(S)$ la valeur optimale de la fonction objectif si on n'ordonne que les tâches de $S \subseteq \mathcal{J}$
 - $f_{\max}^*(\mathcal{J}) \geq \min\{f_j(P), j = 1, \dots, n\}$
 - $f_{\max}^*(\mathcal{J}) \geq f_{\max}^*(\mathcal{J} - \{J_j\})$
 - La procédure "Least-Cost-Last" est optimale pour $1||f_{\max}$
 - Le problème $1|prec|f_{\max}$ est-t-il polynomial?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour

résoudre les
problèmes

NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

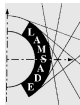
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Problème $1||f_{\max}$

- On note $P = p_1 + p_2 + \dots + p_n$
- J_k la tâche telle que $f_k(P) = \min\{f_j(P), j = 1, \dots, n\}$
- Montrons qu'il \exists un ordonnancement optimal tel que J_k soit la dernière tâche exécutée
- $f_{\max}^*(S)$ la valeur optimale de la fonction objectif si on n'ordonne que les tâches de $S \subseteq \mathcal{J}$
- $f_{\max}^*(\mathcal{J}) \geq \min\{f_j(P), j = 1, \dots, n\}$
- $f_{\max}^*(\mathcal{J}) \geq f_{\max}^*(\mathcal{J} - \{J_j\})$
- La procédure "Least-Cost-Last" est optimale pour $1||f_{\max}$
- Le problème $1|prec|f_{\max}$ est-t-il polynomial?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

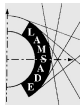
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J|C_{\max}$

Problème $1||f_{\max}$

- On note $P = p_1 + p_2 + \dots + p_n$
- J_k la tâche telle que $f_k(P) = \min\{f_j(P), j = 1, \dots, n\}$
- Montrons qu'il \exists un ordonnancement optimal tel que J_k soit la dernière tâche exécutée
- $f_{\max}^*(S)$ la valeur optimale de la fonction objectif si on n'ordonne que les tâches de $S \subseteq \mathcal{J}$
- $f_{\max}^*(\mathcal{J}) \geq \min\{f_j(P), j = 1, \dots, n\}$
- $f_{\max}^*(\mathcal{J}) \geq f_{\max}^*(\mathcal{J} - \{J_j\})$
- La procédure "Least-Cost-Last" est optimale pour $1||f_{\max}$
- Le problème $1|prec|f_{\max}$ est-t-il polynomial?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

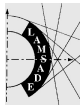
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème $1||f_{\max}$

- On note $P = p_1 + p_2 + \dots + p_n$
- J_k la tâche telle que $f_k(P) = \min\{f_j(P), j = 1, \dots, n\}$
- Montrons qu'il \exists un ordonnancement optimal tel que J_k soit la dernière tâche exécutée
- $f_{\max}^*(S)$ la valeur optimale de la fonction objectif si on n'ordonne que les tâches de $S \subseteq \mathcal{J}$
- $f_{\max}^*(\mathcal{J}) \geq \min\{f_j(P), j = 1, \dots, n\}$
- $f_{\max}^*(\mathcal{J}) \geq f_{\max}^*(\mathcal{J} - \{J_j\})$
- La procédure "Least-Cost-Last" est optimale pour $1||f_{\max}$
- Le problème $1|prec|f_{\max}$ est-t-il polynomial ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

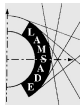
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème $1||f_{\max}$

- On note $P = p_1 + p_2 + \dots + p_n$
- J_k la tâche telle que $f_k(P) = \min\{f_j(P), j = 1, \dots, n\}$
- Montrons qu'il \exists un ordonnancement optimal tel que J_k soit la dernière tâche exécutée
- $f_{\max}^*(S)$ la valeur optimale de la fonction objectif si on n'ordonne que les tâches de $S \subseteq \mathcal{J}$
- $f_{\max}^*(\mathcal{J}) \geq \min\{f_j(P), j = 1, \dots, n\}$
- $f_{\max}^*(\mathcal{J}) \geq f_{\max}^*(\mathcal{J} - \{J_j\})$
- La procédure "Least-Cost-Last" est optimale pour $1||f_{\max}$
- Le problème $1|prec|f_{\max}$ est-t-il polynomial ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

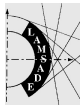
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème 1|| $\sum U_j$

n tâches à ordonnancer telles que $d_1 \leq d_2 \leq \dots \leq d_n$

Algorithme de Hodgson

- 1 : $C = 0, \mathcal{O} \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$
- 2 : **for** $j = 1$ **to** n **do**
- 3 : $C \leftarrow C + p_j$
- 4 : $\mathcal{O} = \mathcal{O} \cup \{J_j\}$
- 5 : **if** $C > d_j$ **then**
- 6 : Soit J_i le job de \mathcal{O} de plus grande durée
- 7 : $\mathcal{O} = \mathcal{O} - \{J_i\}$, $\mathcal{L} = \mathcal{L} \cup \{J_i\}$
- 8 : $C = C - p_j$
- 9 : **end if**
- 10 : **end for**
- 11 : Ordonnancer les jobs de \mathcal{O} dans l'ordre naturel puis les jobs de \mathcal{L} dans n'importe quel ordre.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problème $1||\sum U_j$

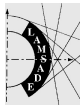
Theorem

L'algorithme de Hodgson est optimal pour $1||\sum U_j$

Exemple :

J_j	J_1	J_2	J_3	J_4	J_5	J_6
p_j	6	4	7	8	3	5
d_j	8	9	15	20	21	22

L'algorithme de Hodgson n'est pas optimal pour $1||\sum w_j U_j$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

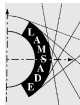
Méthodes constructives

Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthodes constructives

- Comment peut résoudre rapidement un problème d'ordonnancement NP-difficile ?

- 1 Algorithmes à base de règle(s) de priorité
- 2 Algorithmes par décomposition spatiale
- 3 Algorithmes par décomposition temporelle
- 4 Algorithmes par décomposition spatiale et temporelle



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

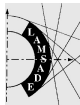
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

$$F||C_{max}$$

Heuristique de Campbell, Dudek, et Smith (CDS)

- Construire des ordonnancements de permutation
- Utiliser l'algorithme de Johnson sur $m - 1$ problèmes $F2||C_{max}$
 \implies une séquence pour chaque problème
- Evaluer les séquences trouvées pour le problème $F||C_{max}$
- Sélectionner la meilleure séquence



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

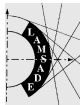
Méthodes constructives

Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{max}$
Problème $J||C_{max}$

$$P|r_j|L_{max}$$

Heuristiques étudiants ID (2004-2005)

- Affecter les tâches aux machines
- Utiliser l'algorithme de Carlier pour résoudre m problèmes $1|r_j|L_{max}$ pour le séquençement sur chaque machine
- une métaheuristique pour améliorer l'affectation



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Problèmes d'optimisation discrète

Definition

Problèmes d'optimisation discrète

- Etant donné un ensemble S de solutions réalisables
 - Etant donnée une fonction cout $c : S \rightarrow IR$
 - Trouver une solution $s^* \in S$ telle que $c(s^*) \leq c(s)$ pour tout $s \in S$.
-
- Les problèmes d'ordonnancement non préemptifs sont des problèmes d'optimisation discrète.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

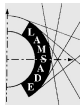
Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthodes de recherche par voisinage

- Démarrer avec une solution initiale s_0
- Passer d'une solution courante s à une autre solution voisine s' (meilleure ou non) tant que le test d'arrêt n'est pas vérifié.
- Le voisinage $N(s)$ d'une solution s est l'ensemble des solutions s' qui peuvent être obtenues à partir de s à travers une modification bien définie.
- Faire une recherche dans le voisinage de s
- Evaluer les solutions $s' \in F(s) \subseteq N(s)$
- Accepter ou rejeter de passer à s' dans l'itération suivante.



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives

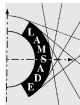
**Heuristiques par
voisinage**

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthodes de recherche par voisinage

Ce qui différencie les méthodes de recherche locale

- La représentation d'une solution
- La conception du voisinage
- La procédure de recherche dans le voisinage
- Le critère d'acceptation ou de rejet d'une solution rencontrée



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

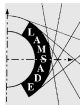
Principe des méthodes B&B

Énoncé du problème

- Soit S l'ensemble des solutions admissibles d'un problème combinatoire \mathcal{P}
- $f(s)$ le critère considéré pour évaluer une solution $s \in S$
- On cherche une solution $s^* \in S$ telle que $f(s^*) = \min_{s \in S} f(s)$

Hypothèses

- On suppose qu'il existe une heuristique permettant de calculer une **borne supérieure** B du minimum recherché : $f(s^*) \leq B$
- On suppose qu'on peut effectuer une **partition** $\{S_1, S_2, \dots, S_k\}$ de S telle que
 - $S_1 \cup S_2 \cup \dots \cup S_k = S$
 - $S_i \cap S_j = \emptyset$ pour tout $i, j \in \{1, \dots, k\}$ et $i \neq j$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

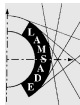
Principe des méthodes B&B

Énoncé du problème

- Soit S l'ensemble des solutions admissibles d'un problème combinatoire \mathcal{P}
- $f(s)$ le critère considéré pour évaluer une solution $s \in S$
- On cherche une solution $s^* \in S$ telle que
$$f(s^*) = \min_{s \in S} f(s)$$

Hypothèses

- On suppose qu'il existe une heuristique permettant de calculer une **borne supérieure** B du minimum recherché :
$$f(s^*) \leq B$$
- On suppose qu'on peut effectuer une **partition** $\{S_1, S_2, \dots, S_k\}$ de S telle que
 - $S_1 \cup S_2 \cup \dots \cup S_k = S$
 - $S_i \cap S_j = \emptyset$ pour tout $i, j \in \{1, \dots, k\}$ et $i \neq j$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

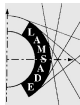
Principe des méthodes B&B

Énoncé du problème

- Soit S l'ensemble des solutions admissibles d'un problème combinatoire \mathcal{P}
- $f(s)$ le critère considéré pour évaluer une solution $s \in S$
- On cherche une solution $s^* \in S$ telle que $f(s^*) = \min_{s \in S} f(s)$

Hypothèses

- On suppose qu'il existe une heuristique permettant de calculer une **borne supérieure** B du minimum recherché : $f(s^*) \leq B$
- On suppose qu'on peut effectuer une **partition** $\{S_1, S_2, \dots, S_k\}$ de S telle que
 - $S_1 \cup S_2 \cup \dots \cup S_k = S$
 - $S_i \cap S_j = \emptyset$ pour tout $i, j \in \{1, \dots, k\}$ et $i \neq j$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

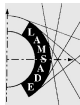
Principe des méthodes B&B

Énoncé du problème

- Soit S l'ensemble des solutions admissibles d'un problème combinatoire \mathcal{P}
- $f(s)$ le critère considéré pour évaluer une solution $s \in S$
- On cherche une solution $s^* \in S$ telle que $f(s^*) = \min_{s \in S} f(s)$

Hypothèses

- On suppose qu'il existe une heuristique permettant de calculer une **borne supérieure** B du minimum recherché : $f(s^*) \leq B$
- On suppose qu'on peut effectuer une **partition** $\{S_1, S_2, \dots, S_k\}$ de S telle que
 - $S_1 \cup S_2 \cup \dots \cup S_k = S$
 - $S_i \cap S_j = \emptyset$ pour tout $i, j \in \{1, \dots, k\}$ et $i \neq j$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

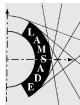
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

Hypothèses (suite)

- On suppose qu'il est possible, pour tout $i = 1, \dots, k$, de calculer une **borne inférieure** b_i de $f(s)$ sur S_i :
$$\forall s \in S_i, b_i \leq f(s)$$

- Si $b_i > B$ alors S_i ne peut contenir une solution optimale
 \Rightarrow Abandonner la recherche dans S_i
- Si $b_i \leq B$ alors S_i peut contenir une solution optimale
 \Rightarrow Continuer la recherche dans S_i
- Si $b_i = B$ alors optimum local



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

Hypothèses (suite)

- On suppose qu'il est possible, pour tout $i = 1, \dots, k$, de calculer une **borne inférieure** b_i de $f(s)$ sur S_i :
$$\forall s \in S_i, b_i \leq f(s)$$

Quelles conclusions peut-on avoir ?

- Si $b_i > B$ alors S_i ne peut contenir une solution optimale
 \Rightarrow Abandonner la recherche dans S_i
- Si $b_i \leq B$ alors S_i peut contenir une solution optimale
 \Rightarrow Continuer la recherche dans S_i
- Si $b_i = B$ alors optimum local



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

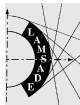
Hypothèses (suite)

- On suppose qu'il est possible, pour tout $i = 1, \dots, k$, de calculer une **borne inférieure** b_i de $f(s)$ sur S_i :
$$\forall s \in S_i, b_i \leq f(s)$$

Quelles conclusions peut-on avoir ?

Pour tout $i = 1, \dots, k$

- Si $b_i > B$ alors S_i ne peut contenir une solution optimale
 \Rightarrow Abandonner la recherche dans S_i
- Si $b_i \leq B$ alors S_i peut contenir une solution optimale
 \Rightarrow Continuer la recherche dans S_i
- Si $b_i = B$ alors optimum local



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

Hypothèses (suite)

- On suppose qu'il est possible, pour tout $i = 1, \dots, k$, de calculer une **borne inférieure** b_i de $f(s)$ sur S_i :
$$\forall s \in S_i, b_i \leq f(s)$$

Quelles conclusions peut-on avoir ?

Pour tout $i = 1, \dots, k$

- Si $b_i > B$ alors S_i ne peut contenir une solution optimale
 \Rightarrow Abandonner la recherche dans S_i
- Si $b_i \leq B$ alors S_i peut contenir une solution optimale
 \Rightarrow Continuer la recherche dans S_i
- Si $b_i = B$ alors optimum local



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

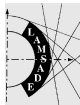
Hypothèses (suite)

- On suppose qu'il est possible, pour tout $i = 1, \dots, k$, de calculer une **borne inférieure** b_i de $f(s)$ sur S_i :
$$\forall s \in S_i, b_i \leq f(s)$$

Quelles conclusions peut-on avoir ?

Pour tout $i = 1, \dots, k$

- Si $b_i > B$ alors S_i ne peut contenir une solution optimale
 \Rightarrow Abandonner la recherche dans S_i
- Si $b_i \leq B$ alors S_i peut contenir une solution optimale
 \Rightarrow Continuer la recherche dans S_i
- Si $b_i = B$ alors optimum local



Plan

Algorithmes à base
de règles de priorité

- Problèmes à une machine
- Problèmes de type flowshop
- Problèmes à machines parallèles
- Limitation des règles de priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

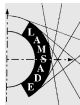
Méthodes constructives
Heuristiques par
voisinage

**Procédures par
évaluation et séparation**

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

Continuer la recherche dans S_i veut dire qu'on effectue une partition de S_i en k_i sous-ensemble $S_{i,1}, S_{i,2}, \dots, S_{i,k_i}$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

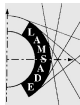
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B

Continuer la recherche dans S_i veut dire qu'on effectue une partition de S_i en k_i sous-ensemble $S_{i,1}, S_{i,2}, \dots, S_{i,k_i}$

Pour tout $j = 1, \dots, k_i$

- Si $b_{i,j} > B$ alors $S_{i,j}$ ne peut contenir une solution optimale
⇒ Abandonner la recherche dans $S_{i,j}$
- Si $b_{i,j} \leq B$ alors $S_{i,j}$ peut contenir une solution optimale
⇒ Continuer la recherche dans $S_{i,j}$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

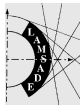
Principe des méthodes B&B

Continuer la recherche dans S_i veut dire qu'on effectue une partition de S_i en k_i sous-ensemble $S_{i,1}, S_{i,2}, \dots, S_{i,k_i}$

Pour tout $j = 1, \dots, k_i$

- Si $b_{i,j} > B$ alors $S_{i,j}$ ne peut contenir une solution optimale
⇒ Abandonner la recherche dans $S_{i,j}$
- Si $b_{i,j} \leq B$ alors $S_{i,j}$ peut contenir une solution optimale
⇒ Continuer la recherche dans $S_{i,j}$

Si pour tout $j = 1, \dots, k_i$, on a abandonné la recherche dans $S_{i,j}$, alors ceci veut dire qu'on abandonne la recherche dans S_i



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

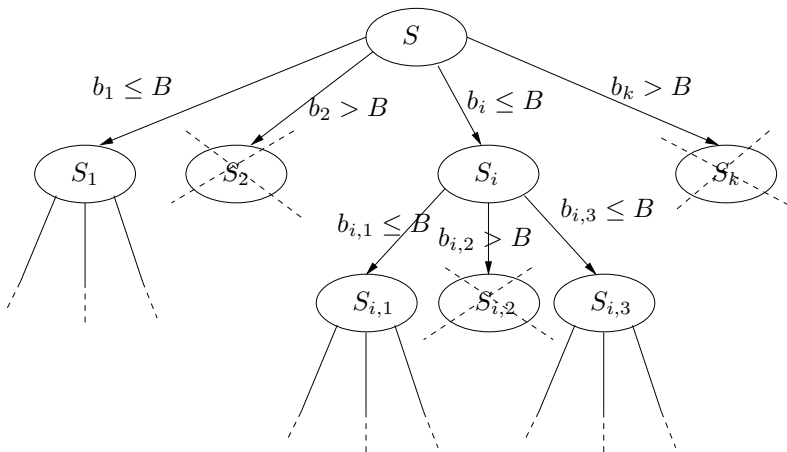
Algorithmes plus
"élaborés"

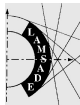
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Principe des méthodes B&B





Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

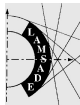
Principe des méthodes B&B

Schémas d'évolution de l'exploration

- En largeur d'abord
- En profondeur d'abord
- Meilleur noeud d'abord (ex : borne inférieure minimale)

Borne supérieure

- Si on trouve dans un noeud donné une borne supérieure $B' < B$, on remplace B par B' dans la suite des calculs
- Le calcul de nouvelles bornes est généralement fait au niveau des feuilles de l'arbre de décision



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

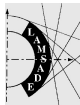
Principe des méthodes B&B

Schémas d'évolution de l'exploration

- En largeur d'abord
- En profondeur d'abord
- Meilleur noeud d'abord (ex : borne inférieure minimale)

Borne supérieure

- Si on trouve dans un noeud donné une borne supérieure $B' < B$, on remplace B par B' dans la suite des calculs
- Le calcul de nouvelles bornes est généralement fait au niveau des feuilles de l'arbre de décision



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

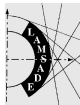
Principe des méthodes B&B

Schémas d'évolution de l'exploration

- En largeur d'abord
- En profondeur d'abord
- Meilleur noeud d'abord (ex : borne inférieure minimale)

Borne supérieure

- Si on trouve dans un noeud donné une borne supérieure $B' < B$, on remplace B par B' dans la suite des calculs
- Le calcul de nouvelles bornes est généralement fait au niveau des feuilles de l'arbre de décision



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

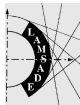
Principe des méthodes B&B

Schémas d'évolution de l'exploration

- En largeur d'abord
- En profondeur d'abord
- Meilleur noeud d'abord (ex : borne inférieure minimale)

Borne supérieure

- Si on trouve dans un noeud donné une borne supérieure $B' < B$, on remplace B par B' dans la suite des calculs
- Le calcul de nouvelles bornes est généralement fait au niveau des feuilles de l'arbre de décision
⇒ la borne est la valeur du critère de l'ordonnement correspondant à la feuille



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

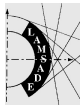
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

... En résumé

Ce qui différencie les approches par évaluation et séparation est :

- le schéma de séparation (comment partitionner S)
- le schéma d'évolution de l'exploration de l'arbre de décision
- les méthodes de calcul des bornes supérieures et inférieures
- la fréquence de calcul des bornes supérieures
- ...



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

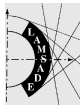
Première méthode

Paramètres de la méthode

■ Schéma de séparation :

- S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier
- $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième
- ...

- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)
- Borne supérieure donnée par l'algorithme de Jackson non-préemptif (on la met à jour à chaque niveau)
- Borne inférieure donnée par l'algorithme de Jackson préemptif
- On construit des ordonnancements actifs !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Paramètres de la méthode

- Schéma de séparation :

- S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier

- $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième

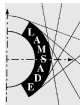
- ...

- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)

- Borne supérieure donnée par l'algorithme de Jackson non-préemptif (on la met à jour à chaque niveau)

- Borne inférieure donnée par l'algorithme de Jackson préemptif

- On construit des ordonnancements actifs !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Paramètres de la méthode

- Schéma de séparation :

- S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier

- $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième

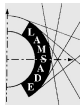
- ...

- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)

- Borne supérieure donnée par l'algorithme de Jackson non-préemptif (on la met à jour à chaque niveau)

- Borne inférieure donnée par l'algorithme de Jackson préemptif

- On construit des ordonnancements actifs !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

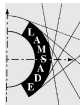
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Paramètres de la méthode

- Schéma de séparation :
 - S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier
 - $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième
 - ...
- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)
 - Borne supérieure donnée par l'algorithme de Jackson non-préemptif (on la met à jour à chaque niveau)
 - Borne inférieure donnée par l'algorithme de Jackson préemptif
 - On construit des ordonnancements actifs !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

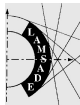
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Paramètres de la méthode

- Schéma de séparation :
 - S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier
 - $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième
 - ...
- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)
- Borne supérieure donnée par l'**algorithme de Jackson non-préemptif** (on la met à jour à chaque niveau)
- Borne inférieure donnée par l'**algorithme de Jackson préemptif**
- On construit des ordonnancements actifs !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

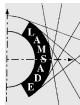
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Paramètres de la méthode

- Schéma de séparation :
 - S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier
 - $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième
 - ...
- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)
- Borne supérieure donnée par l'**algorithme de Jackson non-préemptif** (on la met à jour à chaque niveau)
- Borne inférieure donnée par l'**algorithme de Jackson préemptif**
- On construit des ordonnancements actifs !!



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

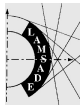
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Paramètres de la méthode

- Schéma de séparation :
 - S_i est l'ensemble des ordonnancements pour lesquels la tâche i est placée en premier
 - $S_{i,j}$ est l'ensemble des ordonnancements pour lesquels i est première position et j en deuxième
 - ...
- Schéma d'évolution de l'exploration de l'arbre de décision : largeur d'abord (par exemple)
- Borne supérieure donnée par l'**algorithme de Jackson non-préemptif** (on la met à jour à chaque niveau)
- Borne inférieure donnée par l'**algorithme de Jackson préemptif**
- **On construit des ordonnancements actifs !!**



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles

Méthodes constructives
Heuristiques par
voisinage

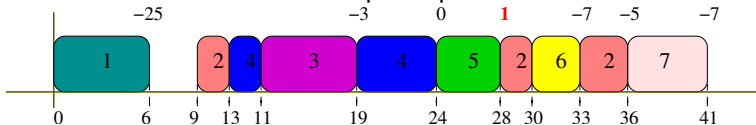
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

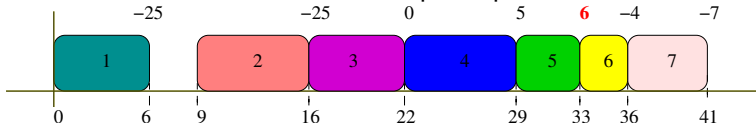
Exemple

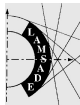
j	1	2	3	4	5	6	7
r_j	0	9	13	11	20	30	30
p_j	6	7	6	7	4	3	5
d_j	31	41	22	24	27	40	48

Ordonnement de Jackson préemptif



Ordonnement de Jackson non-préemptif





Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

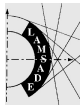
Première méthode

Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

■ Tâches candidates pour la première place :

- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

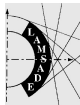
Première méthode

Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

1^{ère} itération

- Tâches candidates pour la première place :
- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

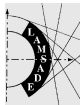
Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

1^{ère} itération

- Tâches candidates pour la première place :

- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine

Problèmes de type
flowshop

Problèmes à machines
parallèles

Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives

Heuristiques par
voisinage

Procédures par
évaluation et séparation

Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

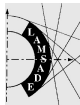
Première méthode

Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

1^{ère} itération

- Tâches candidates pour la première place : J_1 **seulement** parce qu'on construit des ordonnancements actifs
- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

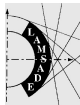
Première méthode

Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

1^{ère} itération

- Tâches candidates pour la première place : J_1 **seulement**
parce qu'on construit des ordonnancements actifs
 $\Rightarrow S_1 = S$: on place J_1 en première position
- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

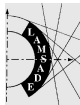
Première méthode

Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

1^{ère} itération

- Tâches candidates pour la première place : J_1 **seulement**
parce qu'on construit des ordonnancements actifs
 $\Rightarrow S_1 = S$: on place J_1 en première position
- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

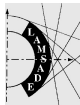
Première méthode

Initialisation

- $t_0 \leftarrow 0$
- $B_0 = 6$ et $b_0 = 1$

1^{ère} itération

- Tâches candidates pour la première place : J_1 **seulement**
parce qu'on construit des ordonnancements actifs
 $\Rightarrow S_1 = S$: on place J_1 en première position
- $b_1 = b_0 = 1$ et $B_1 = B_0 = 6 \Rightarrow$ on poursuit la recherche
- $t_1 \leftarrow \max(r_1, t_0) + p_1 = 6$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élabrés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

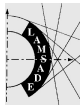
Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place :

- $b_{1,2} = 6 ; b_{1,3} = 3 ; b_{1,4} = 2$

- $B_{1,2} = B_1 = 6 ; B_{1,3} = 3 ; B_{1,4} = 2$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

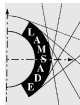
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

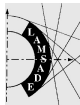
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$

$$r^* = (1, 4, 2)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

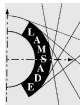
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$

- $\pi^* = (1, 4, \pi)$
- π est la séquence donnée par l'algorithme de Jackson non-préemptif appliqué sur l'instance composée des tâches $\{J_2, J_3, J_5, J_6, J_7\}$ à partir de l'instant $t = \max(t_1, r_4) + p_4 = 18$
- $\pi^* = (1, 4, 3, 5, 6, 2, 7)$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

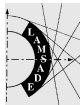
Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$

⇒ On a trouvé une solution optimale

- $\pi^* = (1, 4, \pi)$
- π est la séquence donnée par l'algorithme de Jackson non-préemptif appliqué sur l'instance composée des tâches $\{J_2, J_3, J_5, J_6, J_7\}$ à partir de l'instant $t = \max(t_1, r_4) + p_4 = 18$
- $\pi^* = (1, 4, 3, 5, 6, 2, 7)$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

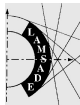
Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$

⇒ On a trouvé une solution optimale

- $\pi^* = (1, 4, \pi)$
 - π est la séquence donnée par l'algorithme de Jackson non-préemptif appliqué sur l'instance composée des tâches $\{J_2, J_3, J_5, J_6, J_7\}$ à partir de l'instant $t = \max(t_1, r_4) + p_4 = 18$
 - $\pi^* = (1, 4, 3, 5, 6, 2, 7)$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

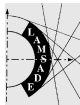
Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$

⇒ On a trouvé une solution optimale

- $\pi^* = (1, 4, \pi)$
- π est la séquence donnée par l'algorithme de Jackson non-préemptif appliqué sur l'instance composée des tâches $\{J_2, J_3, J_5, J_6, J_7\}$ à partir de l'instant $t = \max(t_1, r_4) + p_4 = 18$
- $\pi^* = (1, 4, 3, 5, 6, 2, 7)$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

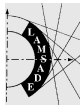
Première méthode

2^{ème} itération

- Tâches candidates pour la deuxième place : J_2 , J_3 et J_4
(respect du fait qu'on construit des ordonnancements actifs)
- $b_{1,2} = 6$; $b_{1,3} = 3$; $b_{1,4} = 2$
- $B_{1,2} = B_1 = 6$; $B_{1,3} = 3$; $B_{1,4} = 2$

⇒ On a trouvé une solution optimale

- $\pi^* = (1, 4, \pi)$
- π est la séquence donnée par l'algorithme de Jackson non-préemptif appliqué sur l'instance composée des tâches $\{J_2, J_3, J_5, J_6, J_7\}$ à partir de l'instant $t = \max(t_1, r_4) + p_4 = 18$
- $\pi^* = (1, 4, 3, 5, 6, 2, 7)$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

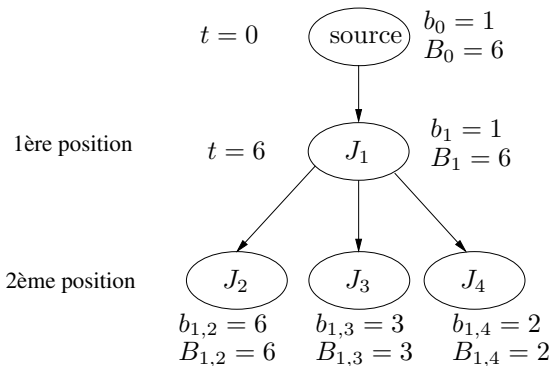
Approches pour
résoudre les
problèmes
NP-difficiles

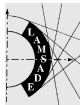
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation

Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Première méthode

Arbre de décision





Plan

Algorithmes à base
de règles de priorité

- Problèmes à une machine
- Problèmes de type flowshop
- Problèmes à machines parallèles
- Limitation des règles de priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

- Méthodes constructives
- Heuristiques par voisinage
- Procédures par évaluation et séparation

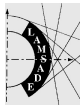
Résolution du problème

$1|r_j|L_{\max}$

Problème $J||C_{\max}$

Deuxième méthode

Voir feuilles jointes



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

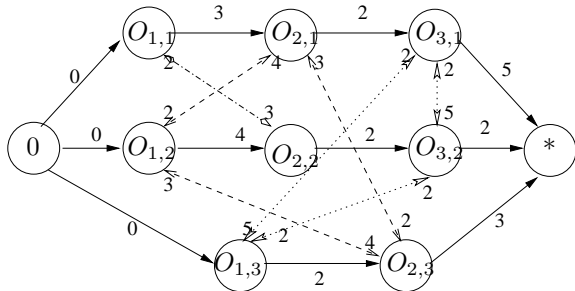
Algorithmes plus
"élaborés"

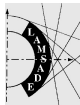
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Modélisation

J_j	J_1			J_2			J_3	
$O_{i,j}$	$O_{1,1}$	$O_{2,1}$	$O_{3,1}$	$O_{1,2}$	$O_{2,2}$	$O_{3,2}$	$O_{1,3}$	$O_{2,3}$
$M_{i,j}$	M_1	M_2	M_3	M_2	M_1	M_3	M_3	M_2
$p_{i,j}$	3	2	5	4	2	2	2	3





Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Modélisation

- Modélisation avec un graphe disjonctif $G = (V, C, D)$
 - V ensemble des sommets : les opérations
 - C ensemble des arcs conjonctifs : contraintes de
précédence entre opérations d'un même job + arcs dûs
aux sommets source (0) et puits (*)
 - D ensemble des "arcs" disjonctifs : conflits entre les
opérations utilisant la même machine
- Résoudre le problème de jobshop consiste à orienter les
arcs disjonctifs **sans créer de circuit**
⇒ ordre de passage des tâches sur une même ressource
- Le plus long chemin dans le graphe résultant détermine la
durée totale de l'ordonnancement



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

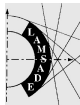
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Modélisation

- Modélisation avec un graphe disjonctif $G = (V, C, D)$
 - V ensemble des sommets : les opérations
 - C ensemble des arcs conjonctifs : contraintes de
précédence entre opérations d'un même job + arcs dûs
aux sommets source (0) et puits (*)
 - D ensemble des "arcs" disjonctifs : conflits entre les
opérations utilisant la même machine
- Résoudre le problème de jobshop consiste à orienter les
arcs disjonctifs **sans créer de circuit**
⇒ ordre de passage des tâches sur une même ressource
- Le plus long chemin dans le graphe résultant détermine la
durée totale de l'ordonnancement



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

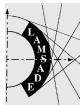
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Modélisation

- Modélisation avec un graphe disjonctif $G = (V, C, D)$
 - V ensemble des sommets : les opérations
 - C ensemble des arcs conjonctifs : contraintes de
précédence entre opérations d'un même job + arcs dûs
aux sommets source (0) et puits (*)
 - D ensemble des "arcs" disjonctifs : conflits entre les
opérations utilisant la même machine
- Résoudre le problème de jobshop consiste à orienter les
arcs disjonctifs **sans créer de circuit**
⇒ ordre de passage des tâches sur une même ressource
- Le plus long chemin dans le graphe résultant détermine la
durée totale de l'ordonnement



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

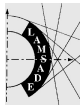
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- A un ordonnancement partiel (ou complet) S est associé un graphe $G_S = (V, C_S, D_S)$
 - $C_S = C \cup CD(S)$, $CD(S)$ est l'ensemble des arcs qui remplacent certains arcs de D (selon S)
 - $D_S = D - DC(S)$, $DC(S)$ est l'ensemble des arcs qui ont été arbitrés (remplacés par les arcs conjonctifs $CD(S)$)



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

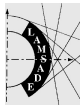
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- A un ordonnancement partiel (ou complet) S est associé un graphe $G_S = (V, C_S, D_S)$
 - $C_S = C \cup CD(S)$, $CD(S)$ est l'ensemble des arcs qui remplacent certains arcs de D (selon S)
 - $D_S = D - DC(S)$, $DC(S)$ est l'ensemble des arcs qui ont été arbitrés (remplacés par les arcs conjonctifs $CD(S)$)



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

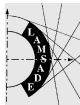
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- A un ordonnancement partiel (ou complet) S est associé un graphe $G_S = (V, C_S, D_S)$
 - $C_S = C \cup CD(S)$, $CD(S)$ est l'ensemble des arcs qui remplacent certains arcs de D (selon S)
 - $D_S = D - DC(S)$, $DC(S)$ est l'ensemble des arcs qui ont été arbitrés (remplacés par les arcs conjonctifs $CD(S)$)



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

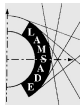
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- Supposons que les ordres de passage ont été décidés sur un sous ensemble de machines $\mathcal{M} \subseteq \{M_1, M_2, \dots, M_m\}$
 - On note $D(k)$ l'ensemble des arcs disjonctifs dûs à la machine M_k : $D = \bigcup_{k=1}^m D(k)$
 - On note $C(k)$ l'ensemble des arcs conjonctifs qui remplacent les arcs disjonctifs $D(k)$ de la machine M_k (pour la solution partielle courante)
 - Le graphe disjonctif correspondant est noté $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$ avec

$$C(\mathcal{M}) = \bigcup_{k \in \mathcal{M}} C(k) \text{ et } D(\mathcal{M}) = \bigcup_{k \notin \mathcal{M}} D(k)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

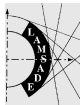
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- Supposons que les ordres de passage ont été décidés sur un sous ensemble de machines $\mathcal{M} \subseteq \{M_1, M_2, \dots, M_m\}$
 - On note $D(k)$ l'ensemble des arcs disjonctifs dûs à la machine M_k : $D = \bigcup_{k=1}^m D(k)$
 - On note $C(k)$ l'ensemble des arcs conjonctifs qui remplacent les arcs disjonctifs $D(k)$ de la machine M_k (pour la solution partielle courante)
 - Le graphe disjonctif correspondant est noté $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$ avec

$$C(\mathcal{M}) = \bigcup_{k \in \mathcal{M}} C(k) \text{ et } D(\mathcal{M}) = \bigcup_{k \notin \mathcal{M}} D(k)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

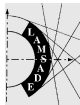
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- Supposons que les ordres de passage ont été décidés sur un sous ensemble de machines $\mathcal{M} \subseteq \{M_1, M_2, \dots, M_m\}$
 - On note $D(k)$ l'ensemble des arcs disjonctifs dûs à la machine M_k : $D = \bigcup_{k=1}^m D(k)$
 - On note $C(k)$ l'ensemble des arcs conjonctifs qui remplacent les arcs disjonctifs $D(k)$ de la machine M_k (**pour la solution partielle courante**)
 - Le graphe disjonctif correspondant est noté $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$ avec

$$C(\mathcal{M}) = \bigcup_{k \in \mathcal{M}} C(k) \text{ et } D(\mathcal{M}) = \bigcup_{k \notin \mathcal{M}} D(k)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

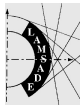
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

- Supposons que les ordres de passage ont été décidés sur un sous ensemble de machines $\mathcal{M} \subseteq \{M_1, M_2, \dots, M_m\}$
 - On note $D(k)$ l'ensemble des arcs disjonctifs dûs à la machine M_k : $D = \bigcup_{k=1}^m D(k)$
 - On note $C(k)$ l'ensemble des arcs conjonctifs qui remplacent les arcs disjonctifs $D(k)$ de la machine M_k (**pour la solution partielle courante**)
 - Le graphe disjonctif correspondant est noté $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$ avec

$$C(\mathcal{M}) = \bigcup_{k \in \mathcal{M}} C(k) \text{ et } D(\mathcal{M}) = \bigcup_{k \notin \mathcal{M}} D(k)$$



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

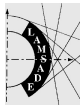
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Principe : La Méthode de *shifting bottleneck* consiste à augmenter l'ensemble \mathcal{M} progressivement jusqu'à ordonnancer toutes les machines

- 1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?
- 2 Comment ordonnancer une machine sélectionnée ?
Quel problème d'ordonnancement considérer sur une machine sélectionnée ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

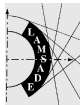
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Principe : La Méthode de *shifting bottleneck* consiste à augmenter l'ensemble \mathcal{M} progressivement jusqu'à ordonnancer toutes les machines

- 1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?
- 2 Comment ordonnancer une machine sélectionnée ?
Quel problème d'ordonnancement considérer sur une machine sélectionnée ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

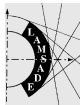
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Principe : La Méthode de *shifting bottleneck* consiste à augmenter l'ensemble \mathcal{M} progressivement jusqu'à ordonnancer toutes les machines

- 1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?
- 2 Comment ordonnancer une machine sélectionnée ?
Quel problème d'ordonnancement considérer sur une machine sélectionnée ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?

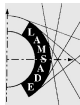
- On suppose qu'un certain nombre de décisions ont été prises, i.e., \mathcal{M} fixé
- On construit un problème d'ordonnancement $P(k, \mathcal{M})$ pour chaque machine $M_k \notin \mathcal{M}$
- On résout chaque problème de façon à optimiser une fonction objectif $f(k, \mathcal{M})$
- On choisit la machine M_{k^*} telle que

$$f(k^*, \mathcal{M}) = \max_{M_k \notin \mathcal{M}} f(k, \mathcal{M})$$

- L'ordonnancement partiel correspond au graphe

$$G(\mathcal{M}) = (V, C(\mathcal{M}) \cup C(k^*), D(\mathcal{M}) - D(k^*))$$

$C(k^*)$ est l'ensemble des arcs dûs à la solution optimale de la machine M_{k^*}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?

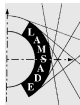
- On suppose qu'un certain nombre de décisions ont été prises, i.e., \mathcal{M} fixé
- On construit un problème d'ordonnancement $P(k, \mathcal{M})$ pour chaque machine $M_k \notin \mathcal{M}$
- On résout chaque problème de façon à optimiser une fonction objectif $f(k, \mathcal{M})$
- On choisit la machine M_{k^*} telle que

$$f(k^*, \mathcal{M}) = \max_{M_k \notin \mathcal{M}} f(k, \mathcal{M})$$

- L'ordonnancement partiel correspond au graphe

$$G(\mathcal{M}) = (V, C(\mathcal{M}) \cup C(k^*), D(\mathcal{M}) - D(k^*))$$

$C(k^*)$ est l'ensemble des arcs dûs à la solution optimale de la machine M_{k^*}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?

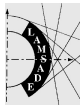
- On suppose qu'un certain nombre de décisions ont été prises, i.e., \mathcal{M} fixé
- On construit un problème d'ordonnancement $P(k, \mathcal{M})$ pour chaque machine $M_k \notin \mathcal{M}$
- On résout chaque problème de façon à optimiser une fonction objectif $f(k, \mathcal{M})$
- On choisit la machine M_{k^*} telle que

$$f(k^*, \mathcal{M}) = \max_{M_k \notin \mathcal{M}} f(k, \mathcal{M})$$

- L'ordonnancement partiel correspond au graphe

$$G(\mathcal{M}) = (V, C(\mathcal{M}) \cup C(k^*), D(\mathcal{M}) - D(k^*))$$

$C(k^*)$ est l'ensemble des arcs dûs à la solution optimale de la machine M_{k^*}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?

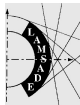
- On suppose qu'un certain nombre de décisions ont été prises, i.e., \mathcal{M} fixé
- On construit un problème d'ordonnancement $P(k, \mathcal{M})$ pour chaque machine $M_k \notin \mathcal{M}$
- On résout chaque problème de façon à optimiser une fonction objectif $f(k, \mathcal{M})$
- On choisit la machine M_{k^*} telle que

$$f(k^*, \mathcal{M}) = \max_{M_k \notin \mathcal{M}} f(k, \mathcal{M})$$

- L'ordonnancement partiel correspond au graphe

$$G(\mathcal{M}) = (V, C(\mathcal{M}) \cup C(k^*), D(\mathcal{M}) - D(k^*))$$

$C(k^*)$ est l'ensemble des arcs dûs à la solution optimale de la machine M_{k^*}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?

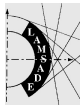
- On suppose qu'un certain nombre de décisions ont été prises, i.e., \mathcal{M} fixé
- On construit un problème d'ordonnancement $P(k, \mathcal{M})$ pour chaque machine $M_k \notin \mathcal{M}$
- On résout chaque problème de façon à optimiser une fonction objectif $f(k, \mathcal{M})$
- On choisit la machine M_{k^*} telle que

$$f(k^*, \mathcal{M}) = \max_{M_k \notin \mathcal{M}} f(k, \mathcal{M})$$

- L'ordonnancement partiel correspond au graphe

$$G(\mathcal{M}) = (V, C(\mathcal{M}) \cup C(k^*), D(\mathcal{M}) - D(k^*))$$

$C(k^*)$ est l'ensemble des arcs dûs à la solution optimale de la machine M_{k^*}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines sont introduites dans \mathcal{M} ?

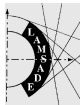
- On suppose qu'un certain nombre de décisions ont été prises, i.e., \mathcal{M} fixé
- On construit un problème d'ordonnancement $P(k, \mathcal{M})$ pour chaque machine $M_k \notin \mathcal{M}$
- On résout chaque problème de façon à optimiser une fonction objectif $f(k, \mathcal{M})$
- On choisit la machine M_{k^*} telle que

$$f(k^*, \mathcal{M}) = \max_{M_k \notin \mathcal{M}} f(k, \mathcal{M})$$

- L'ordonnancement partiel correspond au graphe

$$G(\mathcal{M}) = (V, C(\mathcal{M}) \cup C(k^*), D(\mathcal{M}) - D(k^*))$$

$C(k^*)$ est l'ensemble des arcs dûs à la solution optimale de la machine M_{k^*}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

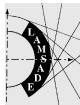
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines ... ?

2 Quel problème d'ordonnancement $P(k, \mathcal{M})$?

- $L(i, j)$ plus long chemin du sommet i au sommet j dans le graphe $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$
- $r_i = L(0, i)$ date de début au plus tôt de l'opération représentée par le sommet i (qui s'exécute sur la machine M_k)
- $d_i = L(0, *) - L(i, *) + p_i$ est la date de fin au plus tard de l'opération représentée par i
- Critère : minimiser le retard algébrique L_{\max} pour la machine M_k sachant que les machines appartenant à \mathcal{M} ont été déjà ordonnancées



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

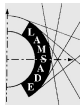
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines ... ?

2 Quel problème d'ordonnancement $P(k, \mathcal{M})$?

- $L(i, j)$ plus long chemin du sommet i au sommet j dans le graphe $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$
- $r_i = L(0, i)$ date de début au plus tôt de l'opération représentée par le sommet i (qui s'exécute sur la machine M_k)
- $d_i = L(0, *) - L(i, *) + p_i$ est la date de fin au plus tard de l'opération représentée par i
- Critère : minimiser le retard algébrique L_{\max} pour la machine M_k sachant que les machines appartenant à \mathcal{M} ont été déjà ordonnancées



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

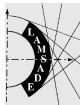
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines ... ?

2 Quel problème d'ordonnancement $P(k, \mathcal{M})$?

- $L(i, j)$ plus long chemin du sommet i au sommet j dans le graphe $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$
- $r_i = L(0, i)$ date de début au plus tôt de l'opération représentée par le sommet i (qui s'exécute sur la machine M_k)
- $d_i = L(0, *) - L(i, *) + p_i$ est la date de fin au plus tard de l'opération représentée par i
- Critère : minimiser le retard algébrique L_{\max} pour la machine M_k sachant que les machines appartenant à \mathcal{M} ont été déjà ordonnancées



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

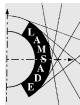
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines ... ?

2 Quel problème d'ordonnancement $P(k, \mathcal{M})$?

- $L(i, j)$ plus long chemin du sommet i au sommet j dans le graphe $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$
- $r_i = L(0, i)$ date de début au plus tôt de l'opération représentée par le sommet i (qui s'exécute sur la machine M_k)
- $d_i = L(0, *) - L(i, *) + p_i$ est la date de fin au plus tard de l'opération représentée par i
- Critère : minimiser le retard algébrique L_{\max} pour la machine M_k sachant que les machines appartenant à \mathcal{M} ont été déjà ordonnancées



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

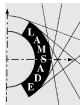
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

1 Dans quel ordre les machines ... ?

2 Quel problème d'ordonnancement $P(k, \mathcal{M})$?

- $L(i, j)$ plus long chemin du sommet i au sommet j dans le graphe $G(\mathcal{M}) = (V, C(\mathcal{M}), D(\mathcal{M}))$
- $r_i = L(0, i)$ date de début au plus tôt de l'opération représentée par le sommet i (qui s'exécute sur la machine M_k)
- $d_i = L(0, *) - L(i, *) + p_i$ est la date de fin au plus tard de l'opération représentée par i
- **Critère** : minimiser le retard algébrique L_{\max} pour la machine M_k sachant que les machines appartenant à \mathcal{M} ont été déjà ordonnancées



Plan

Algorithmes à base
de règles de priorité

- Problèmes à une machine
- Problèmes de type flowshop
- Problèmes à machines parallèles
- Limitation des règles de priorité

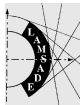
Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

- Méthodes constructives
- Heuristiques par voisinage
- Procédures par évaluation et séparation
- Résolution du problème $1|r_j|L_{\max}$
- Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Pourquoi ce problème ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élabérés"

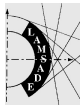
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Pourquoi ce problème ?

- Si les machines n'appartenant pas à \mathcal{M} pouvaient exécuter une infinité d'opérations à la fois, $L(0, *)$ serait le makespan, étant donné l'ordre de passage des opérations sur les machines appartenant à \mathcal{M}



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

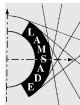
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Pourquoi ce problème ?

- Si les machines n'appartenant pas à \mathcal{M} pouvaient exécuter une infinité d'opérations à la fois, $L(0, *)$ serait le makespan, étant donné l'ordre de passage des opérations sur les machines appartenant à \mathcal{M}
 $\Rightarrow L(0, i)$ est bien la date de début au plus tôt de i



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

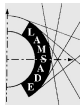
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Pourquoi ce problème ?

- Si les machines n'appartenant pas à \mathcal{M} pouvaient exécuter une infinité d'opérations à la fois, $L(0, *)$ serait le makespan, étant donné l'ordre de passage des opérations sur les machines appartenant à \mathcal{M}
 - $\Rightarrow L(0, i)$ est bien la date de début au plus tôt de i
 - $\Rightarrow L(i, *) - p_i$ est bien le temps nécessaire pour atteindre $*$ lorsque i est terminée



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

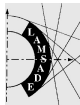
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Pourquoi ce problème ?

- Si les machines n'appartenant pas à \mathcal{M} pouvaient exécuter une infinité d'opérations à la fois, $L(0, *)$ serait le makespan, étant donné l'ordre de passage des opérations sur les machines appartenant à \mathcal{M}
 - $\Rightarrow L(0, i)$ est bien la date de début au plus tôt de i
 - $\Rightarrow L(i, *) - p_i$ est bien le temps nécessaire pour atteindre $*$ lorsque i est terminée
 - \Rightarrow pour ne pas augmenter le makespan, il faut finir i avant $L(0, *) - L(i, *) + p_i$, sinon le makespan augmente au moins d'une quantité égale au retard dû à l'introduction de la machine M_k



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

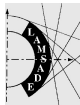
Approches pour
résoudre les
problèmes
NP-difficiles

Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Raffinement

A chaque fois qu'on introduit une nouvelle machine k^* dans \mathcal{M} , on résout les problèmes $P(k, \mathcal{M} - \{k\})$ pour effectuer une optimisation locale



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes
NP-difficiles

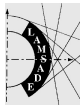
Méthodes constructives
Heuristiques par
voisinage
Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Raffinement

A chaque fois qu'on introduit une nouvelle machine k^* dans \mathcal{M} , on résout les problèmes $P(k, \mathcal{M} - \{k\})$ pour effectuer une optimisation locale

Pourquoi ?



Plan

Algorithmes à base
de règles de priorité

Problèmes à une
machine
Problèmes de type
flowshop
Problèmes à machines
parallèles
Limitation des règles de
priorité

Algorithmes plus
"élaborés"

Approches pour
résoudre les
problèmes

NP-difficiles
Méthodes constructives
Heuristiques par
voisinage

Procédures par
évaluation et séparation
Résolution du problème
 $1|r_j|L_{\max}$
Problème $J||C_{\max}$

Méthode de *shifting bottleneck*

Raffinement

A chaque fois qu'on introduit une nouvelle machine k^* dans \mathcal{M} , on résout les problèmes $P(k, \mathcal{M} - \{k\})$ pour effectuer une optimisation locale

Pourquoi ?

Le problème $P(k, \mathcal{M} - \{k\})$ est différent du problème $P(k, \mathcal{M} \cup \{k^*\} - \{k\})$