Master Académique, Domaine MI, Filière : Mathématiques

Spécialité : Recherche Opérationnelle - Modèles et Méthodes pour l'Ingénierie et le Recherche (RO-2MIR)

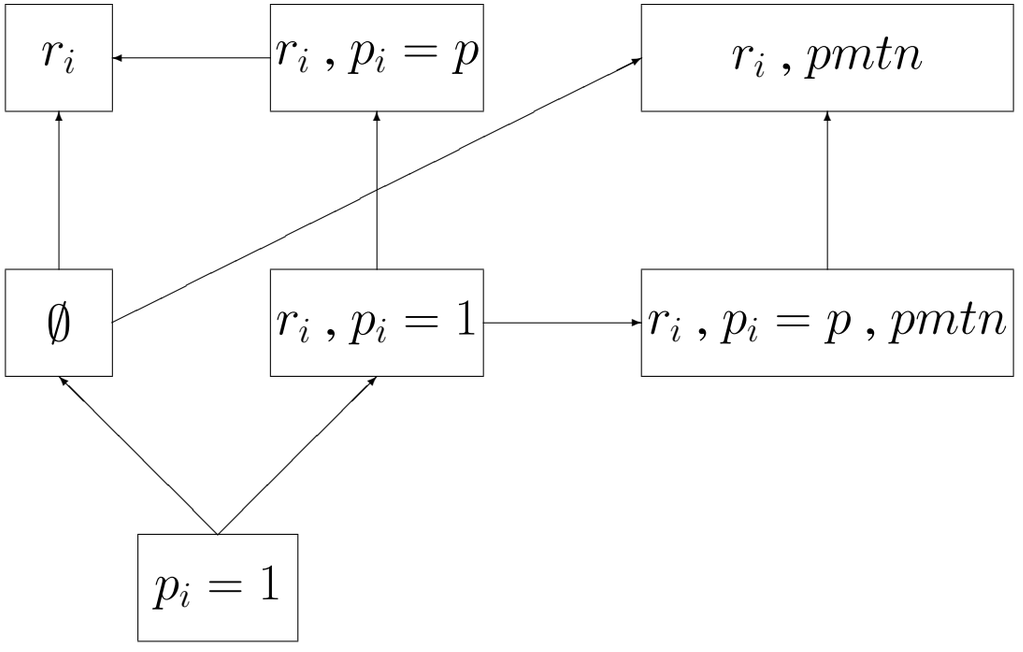
Cours d'Ordonnancement

(2020/2021)

Chapitre 2 :
Ordonnancement sur une seule machine

1 Réductions

Quelques transformations polynomiales



2 Ordonnancement sans dates échues

$\mathbf{1//C_{max}}$

Le problème $1//C_{max}$ est résolu en $O(n)$ avec $C_{max} = \sum_{i=1}^{n} p_i$ (n'importe quelle séquence de tâches, calée à gauche et sans temps mort, fournie une solution optimale). L'algorithme suivant calcule les dates de début de traitement de chaque tâche en les prenant dans l'ordre $1, \ldots, n$.

Algorithme $1//C_{max}$;
début - $t_1 := 0$;
- $C_1 := t_1 + p_1$;
- pour i :=2 haut n
faire - $t_i := C_{i-1}$;
- $C_i := t_i + p_i$
fait ;
- $C_{max} := C_n$
fin

t_i étant la date de début de traitement de la tâche T_i.

le même algorithme peut être aussi utilisé pour résoudre ce problème. La préemption n'apporte aucune amélioration.

$\mathbf{1/r_i/C_{max}}$

Théorème 1 *Le problème* $1/r_i/C_{max}$ *est résolu en* $O(nlogn)$ *en rangeant les tâches par ordre croissant de leurs dates de disponibilité.*

L'algorithme suivant calcule les dates de début de traitement de chaque tâche.

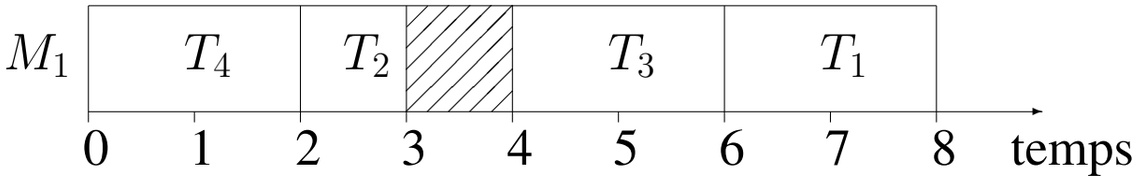
Algorithme $1/r_i/C_{max}$;
début - Ranger les tâches dans l'ordre croissant de leurs dates de disponibilité (soit $1, \ldots, n$) ;
- $t_1 := r_1$;
- $C_1 := t_1 + p_1$;
- pour i :=2 haut n
faire - $t_i := \max\{C_{i-1}, r_i\}$;
- $C_i := t_i + p_i$
fait ;
- $C_{max} := C_n$

fin

Exemple 1 *On désire traiter 4 tâches T_1, T_2, T_3, T_4 dont les temps de traitement et les dates de disponibilité sont données ci-dessous.*

T_i	T_1	T_2	T_3	T_4
p_i	2	1	2	2
r_i	6	1	4	0

Une solution optimale est obtenue en rangeant les tâches dans ordre croissant de leurs dates de disponibilité, on obtient : T_4 , T_2 , T_3 , T_1. Elle est donnée à la figure suivante avec $C_{max} = 8$.



$1/\text{set-up}/C_{max}$

Supposons qu'il existe un temps de transition (set-up time) $c_{ij} = c(T_i, T_j)$ entre deux tâches T_i et T_j et qui correspond, en pratique, au temps de réglage, d'ajustement ou de changement d'outil.

Le problème $1/\text{set-up}/C_{max}$ est équivalent au problème du chemin hamiltonien dans un graphe complet $G = (V, E)$ où V est l'ensemble des tâches et où chaque arête reliant deux tâches T_i et T_j est valuée c_{ij}.

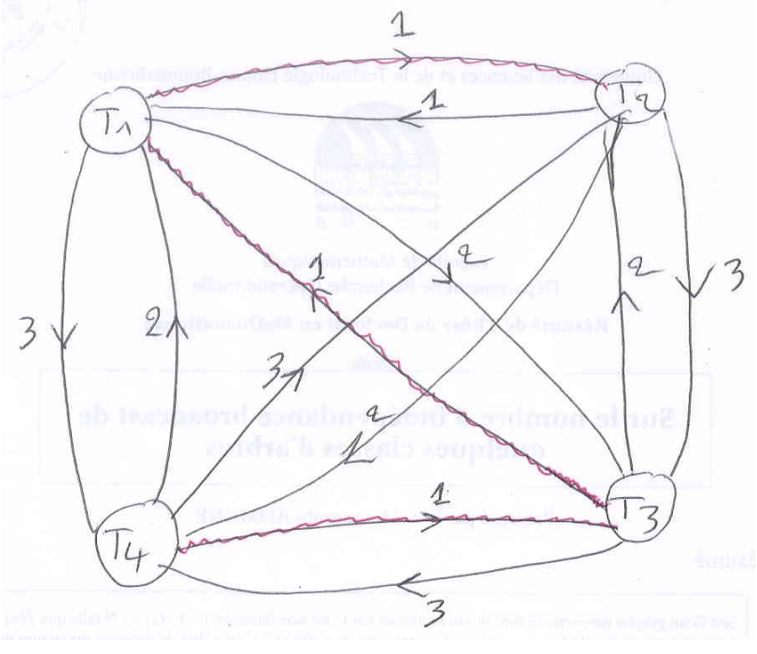
Théorème 2 *Ce problème est NP-difficile.*

Exemple 2 *On désire traiter 4 tâches T_1, T_2, T_3, T_4 dont les temps de traitement et les temps de transitions sont données ci-dessous.*

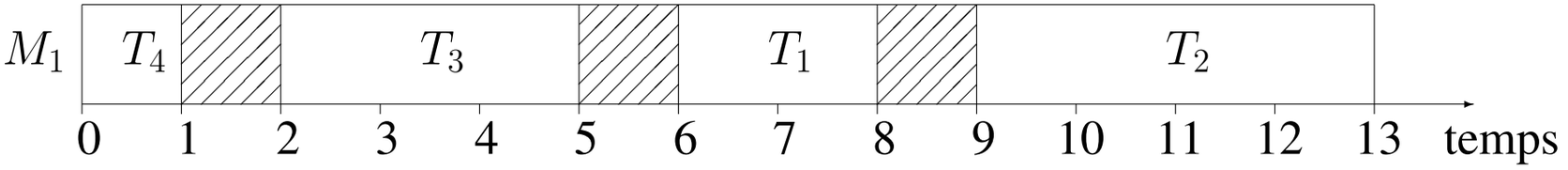
T_i	T_1	T_2	T_3	T_4
p_i	2	4	3	1

c_{ij}	T_1	T_2	T_3	T_4
T_1	0	1	2	3
T_2	1	0	3	2
T_3	1	2	0	3
T_4	2	3	1	0

Le graphe correspondant est



La solution optimale est donnée ci-dessous avec $C_{max} = 13$.



Elle correspond au chemin hamiltonien T_4, T_3, T_1, T_2 dans le graphe correspondant.

$\mathbf{1/r_i, \tilde{d}_i/C_{max}}$

Théorème 3 *Ce problème est NP-difficile.*

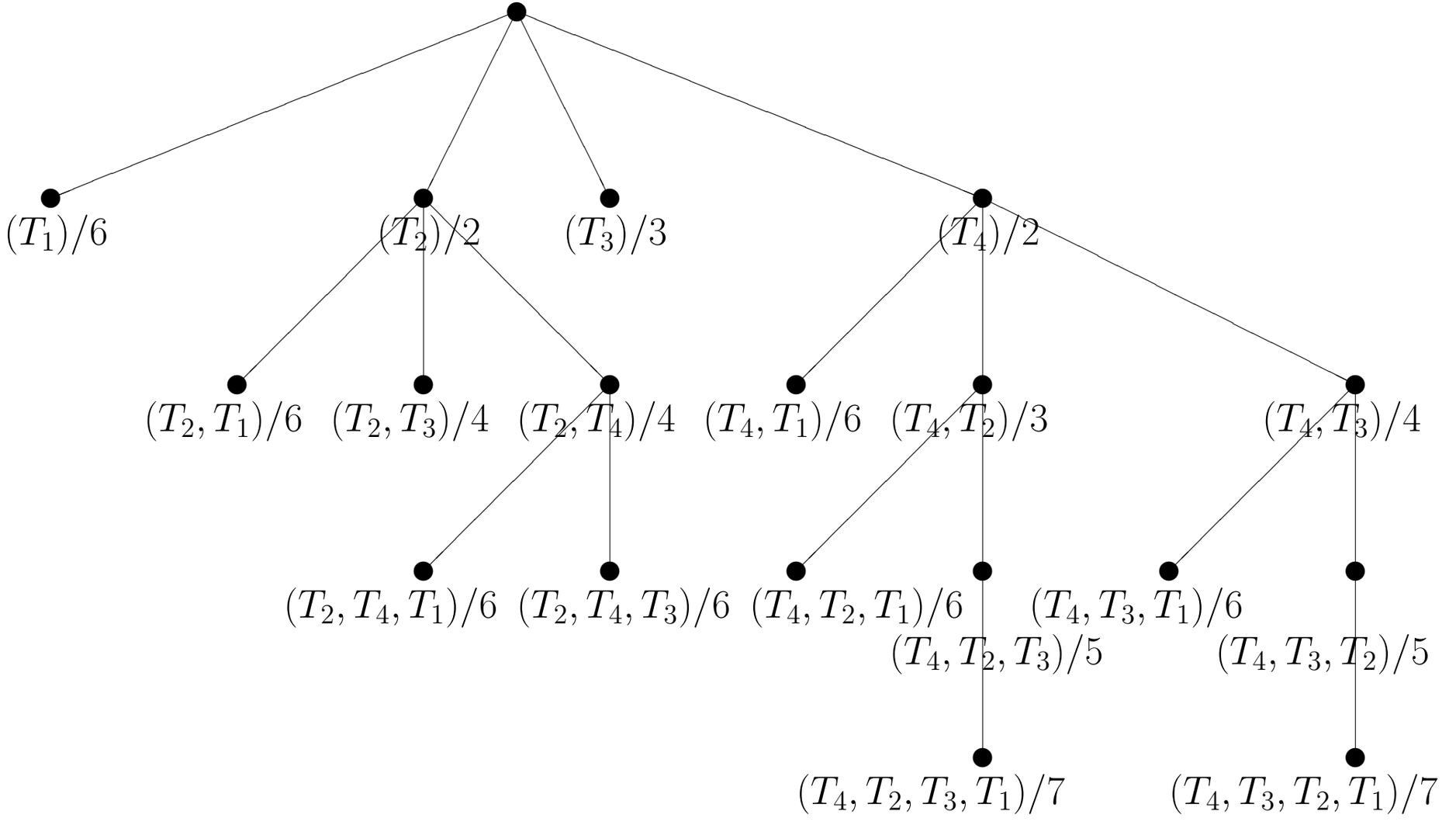
Une solution algorithmique consiste à énumérer tous les ordonnancements possibles et choisir le meilleur, on peut aussi améliorer cet technique en énumérant de façon intelligente : tous les ordonnancements possibles sont implicitement énumérés par la construction d'une arborescence de recherche. A la racine, aucune tâche n'est affectée. Au premier niveau, le noeud numéro i ($i = 1, \ldots, n$), concerne l'ordonnancement de la tâche T_i qui est traitée en première position entre la date r_i et la date $r_i + p_i$. De manière générale, Chaque noeud d'un niveau i ($i = 1, \ldots, n-1$) est connecté à $n - i$ noeuds au niveau $i + 1$. Chaque noeud j ($j = 1, \ldots, n - i$) concerne l'ordonnancement de la tâche numéro j de la liste des tâches non encore ordonnancées. L'arborescence est parcourue en profondeur d'abord.

Si la date de fin de traitement d'une tâche excède sa date limite alors le noeud n'est pas considéré.

Exemple 3 *Soit à ordonnancer 4 tâches $T_1, \ldots, T_4$ dont les dates de disponibilité, les temps de traitement et les dates limites sont donnés dans le tableau ci-dessous.*

T_i	T_1	T_2	T_3	T_4
r_i	4	1	1	0
p_i	2	1	2	2
$\tilde{d}_i$	7	5	6	4

L'arborescence construite est la suivante :


$(T_1)/6$
$(T_2)/2$
$(T_3)/3$
$(T_4)/2$
$(T_2, T_1)/6$
$(T_2, T_3)/4$
$(T_2, T_4)/4$
$(T_4, T_1)/6$
$(T_4, T_2)/3$
$(T_4, T_3)/4$
$(T_2, T_4, T_1)/6$
$(T_2, T_4, T_3)/6$
$(T_4, T_2, T_1)/6$
$(T_4, T_3, T_1)/6$
$(T_4, T_2, T_3)/5$
$(T_4, T_3, T_2)/5$
$(T_4, T_2, T_3, T_1)/7$
$(T_4, T_3, T_2, T_1)/7$

La séquence optimale est T_4 , T_2 , T_3 , T_1 de coût $C_{max} = 7$. Le chiffre après la parenthèse fermante indique la date de fin de traitement des tâches ordonnancées : pour la séquence (T_2, T_1) du deuxième niveau nous avons $c_1 = max\{c_2, r_1\} + p_1 = max\{2, 4\} + 2 = 6$.

Dans le cas où les temps de traitement des tâches sont tous égaux à 1, il existe un algorithme polynomial pour résoudre le problème $1/r_i, p_i = 1, \tilde{d}_i/C_{max}$.

Le problème $1/pmtn, r_i, \tilde{d}_i/C_{max}$ peut être formulé comme un problème de flot maximum et peut donc être résolu en temps polynomial.

$1/r_i$,delivery times$/C_{max}$

Supposons qu'il existe, en plus du temps de traitement, un temps de finition ou de livraison sur d'autres machines. Ce temps de traitement secondaire noté q_i constitue la deuxième phase de traitement des tâches.

Théorème 4 *Ce problème est NP-difficile.*

L'heuristique suivante basée sur l'idée qu'une tâche de plus grand temps de finition est choisie parmi les tâches de plus petite date de disponibilité.

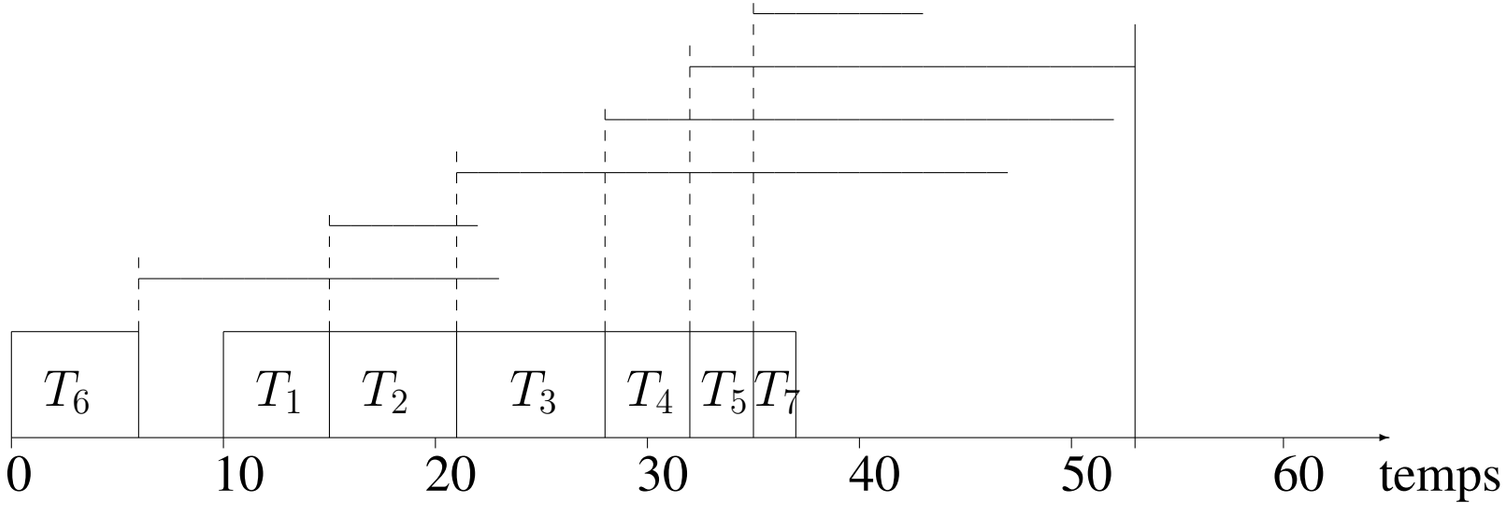
Algorithme $1/r_i$,delivery times$/C_{max}$;
début - $t := \min_{T_i \in T}\{r_i\}$;
- tantque $T \neq \emptyset$
faire - $T' := \{T_i \in T / r_i \leq t\}$;
- Choisir $T_i \in T'$ telle que $p_i = \max_{T_k \in T'}\{p_k / q_k = \max_{T_l \in T'}\{q_l\}\}$;
- Ordonnancer T_i à l'instant t ; $C_i := t + p_i$;
- $T := T \setminus \{T_i\}$;
- $t := \max\{C_i, \min_{T_l \in T, T \neq \emptyset}\{r_l\}\}$;
fait ;
- $C_{max} := \max_{1 \leq i \leq n}\{C_i + q_i\}$
fin

Exemple 4 *Soit à ordonnancer 7 tâches $T_1, \ldots, T_7$ dont les dates de disponibilité, les temps de traitement et les temps de finition sont donnés dans le tableau ci-dessous.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7
r_i	10	13	11	20	30	0	30
p_i	5	6	7	4	3	6	2
q_i	7	26	24	21	8	17	0

La séquence obtenue par l'algorithme est : T_6 , T_1 , T_2 , T_3 , T_4 , T_5 , T_7 de durée 53.

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7
t_i	10	15	21	28	32	0	35
C_i	15	21	28	32	35	6	37
$C_i + q_i$	22	47	52	53	43	23	37



La séquence optimale est : T_6 , T_3 , T_2 , T_4 , T_1 , T_5 , T_7 de durée $C_{max} = 50$.

$1//\overline{C_w}$

La règle SPT (Shortest Processing Time) range les tâches dans l'ordre croissant de leurs temps de traitement et la règle WSPT (Weighted Shortest Processing Time) range les tâches de sorte que $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_n}{w_n}$.

Théorème 5 *La règle WSPT résout le problème* $1//\overline{C_w}$ *en* $O(nlogn)$.

Corollaire 6 *La règle SPT résout le problème* $1//\overline{C}$ *en* $O(nlogn)$.

Preuve. Il suffit de prendre $w_1 = w_2 = \ldots = w_n = 1$. $\square$

Exemple 5 *Soit à ordonnancer 10 tâches $T_1, \ldots, T_{10}$ dont les temps de traitement et les poids sont donnés dans le tableau 2.1 ci-dessous.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_i	16	12	19	4	7	11	12	10	6	8
w_i	2	4	3	2	5	5	1	3	6	2

La solution optimale est obtenue en rangeant les tâches suivant la règle WSPT.
On obtient la liste des tâches $(T_9, T_5, T_4, T_6, T_2, T_8, T_{10}, T_3, T_1, T_7)$
qui correspond à $\frac{6}{6} < \frac{7}{5} < \frac{4}{2} < \frac{11}{5} < \frac{12}{4} < \frac{10}{3} < \frac{8}{2} < \frac{19}{3} < \frac{16}{2} < \frac{12}{1}$
avec $\overline{C_w} = (6.6 + 13.5 + 17.2 + 28.5 + 40.4 + 50.3 + 58.2 + 77.3 + 93.2 + 105.1)/33 = 1223/33 = 37.061$.

Notons que toute permutation de tâches dans cet ordonnancement optimal augmente la valeur de $\overline{C_w}$.

$\mathbf{1/r_i/\overline{C}}$

Ce problème $1/r_i/\overline{C}$ est très difficile à résoudre. Différentes heuristiques ont été proposées dans la littérature.