Ordonnancement et équité

Rapport de stage

Master d'informatique M2

Faculté de Sciences et Ingénieurie de Sorbonne Université

Marion Caumartin

6 mars 2019

Table des matières

1 Introduction

Les problèmes d'ordonnancement constituent un domaine important en recherche opérationnelle. Ils traitent de l'affectation de tâches à des machines, et de l'exécution de ces tâches au cours du temps : il s'agit de savoir où - sur quelle machine - et quand commencera chaque tâche. Récemment, des problèmes d'ordonnancement en présence de différents acteurs ont été étudiés. Cependant, très peu de critères d'équité ont été appliqués aux problèmes d'ordonnancement (hormis le critère classique qui consiste à minimiser le coût maximal d'une machine).

La théorie du partage équitable est un domaine central en choix social computationnel. Elle s'intéresse à définir des règles d'affectation de ressources à des agents de manière à satisfaire au mieux les agents, sachant que les préférences de ceux-ci peuvent avoir différentes structures inhérentes au problème. Un concept fréquemment utilisé pour évaluer les affectations est celui de l'absence d'envie (envy freeness) : une affectation de ressources est dite sans-envie si aucun des agents ne préfère le lot de ressources d'un autre agent à son propre lot.

Le but de ce stage est d'appliquer des concepts et techniques développés en théorie du partage équitable, au domaine de l'ordonnancement. On cherche ainsi à évaluer la qualité des ordonnancements du point de vue de l'équité et à calculer des ordonnancements "équitables".

2 Définition du problème

2.1 Ordonnancement

On note m le nombre de machines.

On note s_i^j le slot de temps i de la machine j.

$\{1, \ldots, n\}$ un ensemble d'agents.

Chaque agent k a un ensemble de $x \in \mathbb{N}$ tâches à réaliser et un critère f_k à minimiser.

On note J_i^k la tâche i de l'agent k.

Chaque tâche J_i^k a une durée p_i^k, peut avoir une deadline d_i^k et une date de disponibilité r_i^k.

Dans un premier temps, on considère que $\forall k \in \{1, \ldots, n\}, \forall i \in \{1, \ldots, x\}, p_i^k = 1$.

Un ordonnancement σ est définit par les dates de début des tâches : $\sigma(J_i^k)$. On note $\mathscr{S}$ l'ensemble des ordonnancements valides. On note $\sigma(k)$ l'ordonnancement de l'agent k pour l'ordonnancement σ.

On note C_i^k la date de fin de la tâche i de l'agent k : $C_i^k = \sigma(J_i^k) + p_i^k$.

$C_{max}^k = \max_i C_i^k$: date de fin maximum (makespan)

$T_i^k = \max(0, C_i^k - d_i^k)$: retard

$L_i^k = C_i^k - d_i^k$: retard algébrique

$U_i^k = \begin{cases} 1 \text{ si } C_i^k > d_i^k \\ 0 \text{ sinon} \end{cases}$: vaut 1 si J_i^k est en retard, 0 sinon

$E_i^k = \max(0, d_i^k - C_i^k)$: avance

$S_i^k = (L_i^k)^2$

$D_i^k = |D_i^k|$

2.2 Equité

On cherche un ordonnancement où chaque agent a exactement x slot : $\forall k \in \{1, \ldots, n\}, |\sigma(k)| = x$.

$\forall k, k' \{1, \ldots, n\}, \sigma(k) \cap \sigma(k') = \emptyset$.

Les slots ne peuvent pas être divisés.

On note $f = (f_1, \ldots, f_n)$.

Pareto-optimalité : σ domine σ' au sens de Pareto ssi
$\begin{cases} \forall k \in \{1, \ldots, n\}, f_k(\sigma(k)) \leq f_k(\sigma'(k)) \\ \exists k \in \{1, \ldots, n\}, f_k(\sigma(k)) < f_k(\sigma'(k)) \end{cases}$

utilitarian social welfare : minimiser $\sum_{k=1}^{n} f_k$.

egalitarian social welfare : minimiser $\max_{k=1}^{n} f_k$.

leximin : $\sigma \succ_{lex} \sigma' \iff \exists i \in \{1, \ldots, n-1\}, \begin{cases} \forall k < i, f_k(\sigma(k)) = f_k(\sigma'(k)) \\ f_i(\sigma(i)) < f_i(\sigma'(i)) \end{cases}$

proportional fair share : on note f_k^* le coût si k disposait des machines pour lui seul.
$\text{pfs}(k) = n f_k^*$.

maxmin fair share : $\text{mfs}(k) = \min_{\sigma \in \mathscr{S}} \max_{i=1}^{n} f_k(\sigma(i))$.

Ordonnancement sans envie : $\forall k, j \in \{1, \ldots, n\}, f_k(\sigma(k)) \leq f_k(\sigma(j))$

Envie entre deux agents : $e_{ij} = \max(0, f_i(\sigma(i)) - f_i(\sigma(j)))$

degré d'envie : $e_i = \max_j e_{ij}$ ou $e_i = \sum_j e_{ij}$

envie de la société : $\max_i e_i$ ou $\sum_i e_i$

Envie up to one slot : i envie j up to one slot ssi $\exists s \in \sigma(i), f_i(\sigma(i) \backslash s) \leq f_i(\sigma(j))$

Envie up to any slot : i envie j up to any slot ssi $\forall s \in \sigma(i), f_i(\sigma(i) \backslash s) \leq f_i(\sigma(j))$

3 Propriétés

Propriété 1. Au moins un ordonnancement egalitarian-optimal est Pareto-optimal.

Démonstration. Par l'absurde :
Soit σ un ordonnancement egalitarian-optimal. Supposons que σ n'est pas Pareto-optimal. Il existe donc un ordonnancement σ' Pareto-optimal qui domine σ et qui n'est pa egalitarian-optimal.
2 cas possibles :

— $\max_{i=1}^{n} f_i(\sigma'(i)) = \max_{i=1}^{n} f_i(\sigma(i))$: comme σ est egalitarian-optimal, alors σ' l'est aussi $\implies$ contradiction.
— $\max_{i=1}^{n} f_i(\sigma'(i)) \leq \max_{i=1}^{n} f_i(\sigma(i))$: donc σ n'est pas egalitarian-optimal $\implies$ contradiction.
□

Propriété 2. Un ordonnancement sans envie n'est pas toujours Pareto-optimal.

Exemple 1. On considère une instance avec une machine et deux agents A et B qui veulent optimiser le même critère. Chaque agent a deux tâches à exécuter sur la machine. On note A_i (reps. B_i) la tâche i de l'agent A (resp. B) pour $i \in \{1, 2\}$. Chaque tâche a une deadline : $d_1^A = 1$, $d_2^A = 3$, $d_1^B = 2$ et $d_2^B = 4$.

On appelle σ_1 l'ordonnancement suivant :

A_1	B_1	B_2	A_2

On appelle σ_2 l'ordonnancement suivant :

A_1	B_1	A_2	B_2

— Si A et B veulent minimiser $\sum T_i$: σ_1 est sans-envie mais n'est pas optimal. En effet, $\begin{cases} f_A(\sigma_1(A)) = 1 \leq f_A(\sigma_1(B)) = 1 \\ f_B(\sigma_1(B)) = 0 \leq f_B(\sigma_1(A)) = 0 \end{cases}$ et $\begin{cases} f_A(\sigma_2(A)) = 0 < f_A(\sigma_1(A)) = 1 \\ f_B(\sigma_2(B)) = 0 \leq f_B(\sigma_1(B)) = 0 \end{cases}$
— Si A et B veulent minimiser $\sum U_i$: σ_1 est sans-envie mais n'est pas optimal. En effet, $\begin{cases} f_A(\sigma_1(A)) = 1 \leq f_A(\sigma_1(B)) = 1 \\ f_B(\sigma_1(B)) = 0 \leq f_B(\sigma_1(A)) = 0 \end{cases}$ et $\begin{cases} f_A(\sigma_2(A)) = 0 < f_A(\sigma_1(A)) = 1 \\ f_B(\sigma_2(B)) = 0 \leq f_B(\sigma_1(B)) = 0 \end{cases}$
— Si A et B veulent minimiser $\sum E_i$: σ_1 est sans-envie mais n'est pas optimal. En effet, $\begin{cases} f_A(\sigma_1(A)) = 0 \leq f_A(\sigma_1(B)) = 0 \\ f_B(\sigma_1(B)) = 1 \leq f_B(\sigma_1(A)) = 1 \end{cases}$ et $\begin{cases} f_A(\sigma_2(A)) = 0 \leq f_A(\sigma_1(A)) = 0 \\ f_B(\sigma_2(B)) = 0 < f_B(\sigma_1(B)) = 1 \end{cases}$
— Si A et B veulent minimiser $\sum S_i$: σ_1 est sans-envie mais n'est pas optimal. En effet, $\begin{cases} f_A(\sigma_1(A)) = 1 \leq f_A(\sigma_1(B)) = 1 \\ f_B(\sigma_1(B)) = 1 \leq f_B(\sigma_1(A)) = 1 \end{cases}$ et $\begin{cases} f_A(\sigma_2(A)) = 0 < f_A(\sigma_1(A)) = 1 \\ f_B(\sigma_2(B)) = 0 < f_B(\sigma_1(B)) = 1 \end{cases}$
— Si A et B veulent minimiser $\sum D_i$: σ_1 est sans-envie mais n'est pas optimal. En effet, $\begin{cases} f_A(\sigma_1(A)) = 1 \leq f_A(\sigma_1(B)) = 1 \\ f_B(\sigma_1(B)) = 1 \leq f_B(\sigma_1(A)) = 1 \end{cases}$ et $\begin{cases} f_A(\sigma_2(A)) = 0 < f_A(\sigma_1(A)) = 1 \\ f_B(\sigma_2(B)) = 0 < f_B(\sigma_1(B)) = 1 \end{cases}$

Propriété 3. Minimiser $\sum L_i$ revient à minimiser $\sum C_i$.

Démonstration. $\sum L_i = \sum (C_i - d_i) = \sum C_i - \sum d_i$. Or $\sum d_i$ est une constante. Donc si on minimise $\sum C_i$, on minimise également $\sum L_i$. □

Exemple 2. On considère une instance avec 2 machines, 3 agents A, B et D qui veulent optimiser $\sum C_i$. Chaque agent a 3 tâches à exécuter. On note A_i (resp. B_i et D_i) la tâche i de l'agent A (reps. B et D) pour $i \in \{1, 2, 3\}$.

On appelle σ_1 l'ordonnancement suivant :

A_1	D_1		A_2	B_3
B_1	D_3	B_2	A_3	D_2

On appelle σ_2 l'ordonnancement suivant :

A_1	D_1	D_2	D_3	B_3
B_1	B_2	A_2	A_3	

σ_1 est sans-envie mais n'est pas optimal. En effet,

$$\begin{cases} f_A(\sigma_1(A)) = 9 \leq f_A(\sigma_1(B)) = 9 \\ f_A(\sigma_1(A)) = 9 \leq f_A(\sigma_1(D)) = 9 \end{cases}, \begin{cases} f_B(\sigma_1(B)) = 9 \leq f_B(\sigma_1(A)) = 9 \\ f_B(\sigma_1(B)) = 9 \leq f_B(\sigma_1(D)) = 9 \end{cases},$$

$$\begin{cases} f_D(\sigma_1(D)) = 9 \leq f_D(\sigma_1(A)) = 9 \\ f_D(\sigma_1(D)) = 9 \leq f_D(\sigma_1(B)) = 9 \end{cases} \text{ et } \begin{cases} f_A(\sigma_2(A)) = 8 < f_A(\sigma_1(A)) = 9 \\ f_B(\sigma_2(B)) = 9 \leq f_B(\sigma_1(B)) = 9 \\ f_D(\sigma_2(D)) = 8 < f_D(\sigma_1(D)) = 9 \end{cases}.$$

4 Algorithme

4.1 Roud-Robin

On considère que tous les agents ont au moins deux tâches.
L'algorithme fonctionne de la façon suivante :

- Chaque agent donne pour chacune de ses tâches un coût aux différents slots.
- On définit un ordre p des agents.
- A chaque tour, chaque agent choisit le slot qui lui coûte le moins.

Exemple 3. On considère une instance avec deux machines et trois agents A, B, et D. Les agents A et B veulent minimiser $\sum T_i$ et l'agent D veut minimiser $\sum C_i$. Chaque agent a trois tâches à executer. On note A_i (resp. B_i et D_i) la tâche i de l'agent A (reps. B et D) pour $i \in \{1, 2, 3\}$. Chaque tâche a une deadline : $d_1^A = 3$, $d_2^A = 1$, $d_2^A = 1$, $d_1^B = 1$, $d_2^B = 2$, $d_3^B = 3$, $d_1^D = 2$, $d_2^D = 4$, et $d_3^D = 4$. On considère l'ordre suivant : DBA. En cas d'égalité des coûts on considère les règles suivantes :

1 on choisit la tâche qui a la plus petite deadline
2 on choisit la tâche qui a l'indice le plus petit
3 on choisit le slot le plus proche de la dealine
4 on choisit la machine avec l'indice le plus petit.

Le tableau des coûts est le suivant :

	s_1^1	s_2^1	s_3^1	s_4^1	s_5^1	s_1^2	s_2^2	s_3^2	s_4^2
A_1	0	0	0	1	2	0	0	0	1
A_2	0	1	2	3	4	0	1	2	3
A_3	0	1	2	3	4	0	1	2	3
B_1	0	1	2	3	4	0	1	2	3
B_2	0	0	1	2	3	0	0	1	2
B_3	0	0	0	1	2	0	0	0	1
D_1	1	2	3	4	5	1	2	3	4
D_2	1	2	3	4	5	1	2	3	4
D_3	1	2	3	4	5	1	2	3	4

Tour 1 :

- Agent D : les slots qui lui coûtent le moins sont s_1^1 et s_1^2 pour chacune de ses tâches. Il choisit donc d'affecter D_1 au slot s_1^1.
- Agent B : les slots qui lui coûtent le moins sont s_1^2 pour B_1, s_2^1, s_1^2 et s_2^2 pour B_2, et s_2^1, s_3^1, s_1^2, s_2^2 et s_3^2 pour B_3. Il choisit donc d'affecter B_1 au slot s_1^2.

— Agent A : les slots qui lui coûtent le moins sont s_2^1, s_3^1, s_2^2 et s_3^2 pour A_1. Il choisit donc d'affecter A_1 au slot s_3^1

D_1		A_1		
B_1				

Tour 2 :

— Agent D : les slots qui lui coûtent le moins sont s_2^1 et s_2^2 pour D_2 et D_3. Il choisit donc d'affecter D_2 au slot s_2^1.

— Agent B : les slots qui lui coûtent le moins sont s_2^2 pour B_2, et s_2^2 et s_3^2 pour B_3. Il choisit donc d'affecter B_2 au slot s_2^2.

— Agent A : le slot qui lui coûtent le moins est s_3^2 pour A_2 et A_3. Il choisit donc d'affecter A_2 au slot s_3^2

D_1	D_2	A_1		
B_1	B_2	A_2		

Tour 3 :

— Agent D : les slots qui lui coûtent le moins sont s_4^1 et s_4^2. Il choisit donc d'affecter D_3 au slot s_4^1.

— Agent B : le slot qui lui coûtent le moins est s_4^2. Il choisit donc d'affecter B_3 au slot s_4^2.

— Agent A : le seul slot restant est s_5^1. Il choisit donc d'affecter A_3 au slot s_5^1.

D_1	D_2	A_1	D_3	A_3
B_1	B_2	A_2	B_3	

L'ordonnancement obtenu est sans-envie up to one slot :

— Agent A : $f_A(\sigma(A)) = 6$, $f_A(\sigma(B)) = 4$ et $f_A(\sigma(D)) = 4$. Mais si on lui retire s_5^1 : $f_A(\sigma(A)\backslash\{s_5^1\}) = 2$.

— Agent B : $f_B(\sigma(B)) = 1$, $f_B(\sigma(A)) = 5$ et $f_B(\sigma(D)) = 1$.

— Agent D : $f_D(\sigma(D)) = 7$, $f_D(\sigma(A)) = 11$ et $f_D(\sigma(B)) = 7$.

Propriété 4. Cet algorithme est sans-envie up to one slot si les coûts sont positifs et que les agents considèrent soit le coût maximal, soit la somme des coûts.

Démonstration. Soit p un ordre sur les agents. Soient $i, j \in \{1, \dots, n\}$. On note s_i^t le slot alloué à l'agent i au tour t, pour $t \in \{1, \dots, x\}$. On note σ l'ordonnancement obtenu avec l'algorithme présenté dans cette partie.

Cas 1 : $i < j$

A chaque tour, $p(i)$ choisi le slot libre le moins coûteux pour lui avant $p(j)$:

$\forall t \in \{1, \dots, x\}, f_{p(i)}(s_{p(i)}^t) \leq f_{p(i)}(s_{p(j)}^t)$

Si i considère le coût maximal : $f_{p(i)}(\sigma(i)) = \max_{t=1}^{x} f_{p(i)}(s_{p(i)}^t) \leq f_{p(i)}(\sigma(j)) = \max_{t=1}^{x} f_{p(i)}(s_{p(j)}^t)$

Si i considère la somme des coûts : $f_{p(i)}(\sigma(i)) = \sum_{t=1}^{x} f_{p(i)}(s_{p(i)}^t) \leq f_{p(i)}(\sigma(j)) = \sum_{t=1}^{x} f_{p(i)}(s_{p(j)}^t)$

Cas 2 : $i > j$

Soit $t \in \{1, \dots, x-1\}$.

Le slot alloué à $p(i)$ au tour t lui coûte moins que le slot alloué à $p(j)$ au tour $t+1$:
$f_{p(i)}(s^t_{p(i)}) \leq f_{p(i)}(s^{t+1}_{p(j)})$
Si i considère le coût maximal :
$f_{p(i)}(\sigma(i)\backslash\{s^x_{p(i)}\}) = \max_{t=1}^{x-1} f_{p(i)}(s^t_{p(i)}) = f_{p(i)}(s^{x-1}_{p(i)}) \leq f_{p(i)}(s^x_{p(j)}) = \max_{t=1}^{x} f_{p(i)}(s^t_{p(j)}) = f_{p(i)}(\sigma(j))$
Si i considère la somme des coûts :
$f_{p(i)}(\sigma(i)\backslash\{s^x_{p(i)}\}) = \sum_{t=1}^{x-1} f_{p(i)}(s^t_{p(i)}) \leq \sum_{t=2}^{x} f_{p(i)}(s^t_{p(j)}) \leq \sum_{t=1}^{x} f_{p(i)}(s^t_{p(j)}) = f_{p(i)}(\sigma(j))$. □