Rapport de projet

Algorithmes de résolution du problème $P_3, h_j|d_i|\sum U_i$

Guillaume Courrier
Carlos Figueiredo
Laure-Anne Poisson
Dany Samy
Axel Vivien

Dans le cadre du cours d'Imed Kacem

Ordonnancement

Feb 2020

Contents

1 Introduction

Le problème $P_3, h_j|d_i|\sum U_i$ correspond à un problème d'ordonnancement des tâches sur 3 machines en parallèle avec les spécificités suivantes :

- contraintes d'indisponibilité des machines définies par les h_j
- contraintes de livraison des tâches définies par les d_i
- chaque tâche a sa proche durée opératoire

L'objectif est de minimiser le nombre de tâche en retard qui est défini par les $\sum U_i$ où $U_i = 1$ si la tâche arrive en retard.

2 Modélisation du problème

2.1 Reformulation

Par son objectif et ses contraintes, il est possible de reformuler différemment le problème :

- à cause de l'indisponibilité, on peut considérer 6 machines au lieu de 3 : les 3 premières représentent chacune une machine avant son indisponibilité (la durée opératoire totale est bornée sur ces machines) et les 3 dernières la machine après (la durée opératoire totale n'est pas bornée)
- à cause de l'objectif : dans le cas des tâches en retard, il est préférable de les mettre après celles qui vont finir à temps. Donc, peu importe la machine sur laquelle on va les exécuter, cela ne va pas impacter l'objectif et, ainsi, on définit une 7ème machine pour les tâches en retard.

Avec ces modifications, l'objectif va consister à minimiser le nombre de tâches à exécuter sur la machine 7.

2.2 Modélisation

On va définir les variables de décisions suivantes :

$$\forall\ (i,j) \in [\![1,n]\!] \times [\![1,7]\!],\ x_{ij} = \begin{cases} 1 & \text{si la tâche } i \text{ est sur la machine } j \\ 0 & \text{sinon.} \end{cases}$$

Par ailleurs, on définit aussi d'autres variables pour nous aider à mieux modéliser notre problème :

- pour tout $j \in [\![1,3]\!]$, T_j correspond au début de l'indisponibilité de la machine j
- pour tout $j \in [\![1,6]\!]$, h_j correspond au surplus lié à l'indisponibilité. En d'autres termes, pour $j \in [\![1,3]\!]$, ce terme vaut 0 alors que, pour $j \in [\![4,6]\!]$, h_j correspond à la fin de d'indisponibilité pour la machine concernée
- une variable M suffisamment grande (par exemple, la somme de toutes les durées opératoires) pour simplifier l'écriture de la contrainte concernant le retard

La modèle linéaire en nombres entiers s'écrit donc :

$$\min \sum_{i=1}^{7} x_{i7}$$

$$sc \sum_{i=1}^{n} P_i * x_{ij} \leq T_j \qquad \forall j \in [\![1,3]\!]$$

$$\sum_{j=1}^{7} x_{ij} = 1 \qquad \forall i \in [\![1,n]\!]$$

$$\sum_{i=1}^{l} P_i * x_{ij} + h_j \leq d_l + (1 - x_{lj}) * M \qquad \forall j \in [\![1,6]\!], \ ,\forall l \in [\![1,n]\!]$$

$$x_{ij} \in \{0,1\} \qquad \forall \ (i,j) \in [\![1,n]\!] \times [\![1,7]\!]$$

3 Heuristiques

Pour toutes les heuristiques, l'affectation des tâches est envisagée dans l'ordre des d_i croissants. On ne place sur M1 à M6 que les tâches qui respectent la date butoir. Les heuristiques diffèrent dans le critère de choix de la machine sur laquelle positionner la tâche sélectionnée. Cette approche étant gloutonne, l'algorithme ne revient jamais sur une décision, si une tâches est ordonnancée dans une machine au départ, elle y restera même si une autre tâche pourrait permettre d'améliorer l'objectif en prenant sa place.

3.1 Heuristique H1

La première heuristique que nous avons développée consiste à ordonnancer les tâches dans la première machine que peut l'exécuter à temps (en respectant les contraintes des machines 1 à 3). Ainsi on teste d'abord si la tâche rentre dans les trois premières machines, puis dans les machines 4, 5, et 6 et enfin, si cette tâche ne peut pas être ordonnancée à temps, on la place dans la machine 7.

3.2 Heuristique H2

Cette heuristique place la tâche sur la machine la plus remplie, telle que la tâche est finie à temps. Ainsi, dans un cas où on rencontrerait une tâche i+1 dont la date butoir est légèrement plus tardive que la tâche i mais dont la durée d'exécution est beaucoup plus longue, on s'assure de lui avoir conservé un plus grand espace. Toutes les machines 1 à 6 sont considérées équitablement, qu'elles représentent une pré-indisponibilité ou pas.

3.3 Heuristique H3

Cette heuristique place la tâche sur la machine la moins remplie, telle que la tâche est finie à temps. Toutes les machines 1 à 6 sont considérées équitablement, qu'elles représentent une pré-indisponibilité ou pas.

3.4 Heuristique H4

La dernière heuristique que nous avons développée consiste à placer la tâche considérée en priorité dans les trois premiers emplacements. On place alors la tâche sur la machine où il y a le plus de place restante avant l'indisponibilité. Si la tâche ne rentre pas, on la place dans les emplacements 4 à 6 après l'indisponibilité, on choisit l'emplacement qui commence le plus tôt. Dans tous les cas, si la tâche ne peut pas finir avant sa date butoir on la place dans l'emplacement 7.

4 Programmation génétique

Cet méthode repose sur une population de solutions, que l'on fait évoluer au cours des itérations de l'algorithme. Le modèle est initialisé à partir d'une génération de solutions viables, qui peut être obtenue en utilisant une des heuristiques programmées. L'idée principale est de générer continuellement de nouvelles solutions en faisant muter nos meilleures anciennes solutions.Nous avons appliqué la démarche suivante :

Pour chaque nouveau membre de la nouvelle génération :

- On choisit un parent selon son score - Nous utilisons un algorithme de roulette pour avoir plus de chances de choisir des parents avec des fits plus élevés (scores de performance).
- On applique une mutation afin de générer la nouvelle solution - Une mutation consiste à déplacer une tache d'une machine vers une autre au hasard.
- On ne garde les nouvelles solutions que si elles sont viables.

Cependant, il existe d'autre variantes possibles, qui pourraient être plus efficaces sur des agencements moins évidents. Par exemple, on pourrait avoir des portions de population complètement inchangées, ou au contraire entièrement nouvelles et aléatoires. Il serait aussi possible d'autoriser des solutions non viables au cours des itérations, en leur affectant un poids faible, puis, de les filtrer à la fin.

5 Résultats

On réalise des tests sur le problème suivant à 5 tâches :
deadlines = {2, 3, 4, 4, 5}
durées = {1, 2, 3, 2, 3}
début d'indisponibilité = {4, 3, 5, 0, 0, 0}
fin d'indisponibilité = {0, 0, 0, 5, 5, 6}

5.1 Simplex

On effectue la résolution avec Cplex. On obtient le résultat suivant :
Schedule: {1, 1, 2, 3, 3}

M1: 1 2 ———
M2: 3 ———
M3: 4 5 ———
Non ordonnancé:

Performance: 0

5.2 Heuristiques

On obtient les ordonnancements suivants :

Heuristique 1	Heuristique 2	Heuristique 3	Heuristique 4
Schedule: {1, 2, 3, 7, 7}	Schedule: {1, 1, 2, 3, 3}	Schedule: {1, 2, 3, 1, 7}	Schedule: {3, 3, 1, 2, 7}
M1: 1 ———	M1: 1 2 ———	M1: 1 4 ———	M1: 3 ———
M2: 2 ———	M2: 3 ———	M2: 2 ———	M2: 4 ———
M3: 3 ———	M3: 4 5 ———	M3: 3 ———	M3: 1 2 ———
Non ordonnancé: 4 5	Non ordonnancé:	Non ordonnancé: 5	Non ordonnancé: 5
Performance: 2	Performance: 0	Performance: 1	Performance: 1

5.3 Algorithme génétique

On part de la solution donnée par l'heuristique 1 et on montre notre meilleure solution après 10000 générations.
Schedule: {2, 2, 1, 3, 3}

M1: 3 ———
M2: 1 2 ———
M3: 4 5 ———
Non ordonnancé:
Performance: 0

6 Conclusion

Ce projet nous a permis de mettre en oeuvre différentes méthodes canoniques pour la résolution des problèmes d'ordonnancement, et d'en comparer les résultats. L'atout du simplexe est sa qualité mais il peut parfois être long. Les heuristiques présentent l'avantage du faible temps d'exécution, toutefois, les solutions qu'elle retournent sont souvent moins efficaces. Elles constituent un bon tandem avec l'algorithme génétique auquel elles peuvent servir de méthode d'initialisation.