Encadré par
Imed Kacem

Projet d'ordonnancement

Problème $P_3, h_j | d_i | \sum U_i$

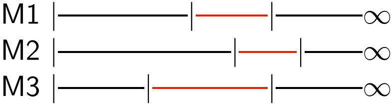
Guillaume Courrier, Carlos Figueiredo, Laure-Anne Poisson
Dany Samy, Axel Vivien

Plan

Présentation du problème

machines j

début d'indisponibilité td_j
fin d'indisponibilité tf_j

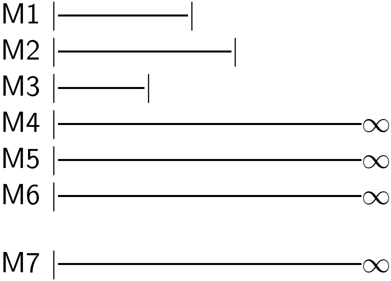


tâches i

durée p_i
deadline d_i
réalisée à temps U_i

=> sortie : affectation de chaque tâche à une machine => objectif : minimser le nombre de tâches en retard

Modélisation



Modélisation linaire en nombre entiers

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

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

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

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

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

Méthodes heuristiques

Pour toutes les heuristiques:
=> les tâches seront envisagées dans l'ordre croissant des d_i
=> on ne place sur M1 à M6 que les tâches qui respectent la deadline

- Placer la tâche avant les indisponibilités si possible, dans l'ordre des machines
- Placer la tâche à t le plus faible
- Placer la tâche à t le plus grand
- Placer la tâche sur la machine où il reste le plus de place

UNIVERSITÉ DE LORRAINE

Algorithme génétique

- Le modèle est initialisé à partir d'une génération de solutions viables - peut être obtenu en utilisant une des heuristiques programmées.

- L'idée principale est de générer continuellement de nouvelles solutions en faisant muter nos meilleures anciennes solutions.

Algorithme génétique

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.

Exemple

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}

Résultats

Résolution sous Cplex

Schedule: {1, 1, 2, 3, 3}
M1: 1 2 |||
M2: 3 |||
M3: 4 5 |||

Non ordonnancé:
-> performance: 0

Résultats

Heuristique 1

Schedule: {1, 2, 3, 7, 7}
M1: 1 |||
M2: 2 |||
M3: 3 |||

Non ordonnancé: 4 5
-> performance: 2

Heuristique 2

Schedule: {1, 1, 2, 3, 3}
M1: 1 2 |||
M2: 3 |||
M3: 4 5 |||

Non ordonnancé:
-> performance: 0

Résultats

Heuristique 3

Schedule: {1, 2, 3, 1, 7}
M1: 1 4 |||
M2: 2 |||
M3: 3 |||

Non ordonnancé: 5
-> performance: 1

Heuristique 4

Schedule: {3, 3, 1, 2, 7}
M1: 3 |||
M2: 4 |||
M3: 1 2 |||

Non ordonnancé: 5
-> performance: 1

Résultats

On part de la solution donnée par l'heuristique 1 et on montre notre meilleure solution après 10000 générations.

Heuristique 1

Schedule: {1, 2, 3, 7, 7}
M1: 1 |||
M2: 2 |||
M3: 3 |||

Non ordonnancé: 4 5
-> performance: 1

Algorithme génétique

Schedule: {2, 2, 1, 3, 3}
M1: 3 |||
M2: 1 2 |||
M3: 4 5 |||

Non ordonnancé:
-> performance: 0

Conclusion

- 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.