Mini projet d'ordonnancement - Projet 7

Valerian Damm Jofrey Luc Quentin Sonrel

18 mai 2017

1 Heuristiques en C

En C, nous représentons le problème par :
— en entrée/contraintes : un tableau de 4 lignes par $nbJobs$ colonnes. Le première ligne indique la date d'arrivée du job, la deuxième son temps d'exécution sur la première machine, la troisième son temps sur la deuxième machine, et la quatrième son temps sur la troisième machine.
— en solution : un tableau de taille $nbJobs$, qui nous donne l'ordre d'exécution des jobs sur les machines. Cet ordre sera le même pour les trois machines, puisque si un ordre est optimal sur la première machine il l'est aussi sur les deux autres.

Nous avons donc cinq fonctions :
— evaluer_solution, qui calcule notre $Cmax$ en prenant en entrée un tableau de contraintes et une solution ;
— heuristique_random, qui nous donne une solution aléatoire (en utilisant l'algorithme Fisher-Yates shuffle) ;
— heuristique_debut_par_somme, qui nous retourne une solution avec les jobs ordonnés par le rapport entre leur date d'arrivée et la somme de leurs temps d'exécution ;
— heuristique_debut, qui nous retourne une solution avec les jobs ordonnés par leur date d'arrivée ;
— heuristique_greedy, qui nous retourne une solution avec les jobs ordonnés par la somme de leurs temps d'exécution.

2 Algorithme génétique

Nous avons fait l'algorithme génétique en java. Le code est séparé entre quatre classes principales :
— Algorithme.java, qui contient l'exécution de l'algorithme génétique en soi (crée une population, boucler sur les générations en sélectionnant les meilleurs individus) ;
— Fitness.java, qui contient la fonction de calcul de la fitness/la qualité d'un individu, c'est à dire le calcul de Cmax dans notre cas.
— Individu.java, qui contient la définition et les fonctions liées à nos individus (en l'occurence des solutions constituées d'un tableau contenant l'ordre d'exécution des jobs) ;
— Population.java, qui contient la définition et les fonctions liées à une population d'individus.

Le main se trouve dans la classe GA.java.

3 Modèle linéaire

Nous n'avons pas réussi à installer CPlex (l'installeur ne se lance même pas et plante au démarrage chez moi), voici donc le modèle linéaire, que nous n'avons pas pu tester. Ce modèle linéaire ne classe les jobs que par rapport à la première machine, puisque l'ordre ne change pas sur les autres.

Fonction objectif : $O = min(max(x_{ij}[debut_j + p_j]))$

Contraintes :

- $O >= \sum_{p_i}$ (l'objectif est forcément supérieur à la somme des temps d'exécution)
- $\sum_{x_{ij}} = nbJobs - 1$ (par exemple, trois variables suffisent à décrire l'ordre de quatre jobs)
- $x_{ii} = 0$
- $x_{ij} \in \{0, 1\}$
- $\sum_{k=p\setminus\{i\}} x_{ik} < 1$ (un job ne précède au maximum qu'un seul autre job)
- $debut_j >= dateArrivee_j$ et $debut_j >= x_{ij}(debut_i + p_i)$ (la date de début d'un job est supérieure à sa date d'arrivée et à la date de fin du job précédent (s'il existe))

Variables :

- $x_{ij} = 1$: le job j suit immédiatement le job i.
- $debut_j$: date de départ sur la première machine de la tâche j.

Données :

- p_j = temps d'éxecutuin de j sur la première machine.
- $dateArrivee_j$ = date minimale de départ de j.

4 Comparaison des résultats

Dans l'ensemble, les heuristiques "simples" codées en C n'étant pas très recherchées, sur les mêmes exemples, l'algorithme génétique donne de meilleurs résultats que les heuristiques. (La meilleure heuristique semblant être celle qui utilise les rapports entre la date d'arrivée des jobs et la somme de leur temps d'exécution).