Recherche opérationnelle

Rapport des TPs 5 et 6 :

LIEGEOIS Marion et MANGÉ Valérian

15 janvier 2021

Table des matières

1 Ordonnancement avec contraintes de précédence

Les codes pour cette partie sont dans le fichier *P1*. Ils s'appuient sur un problème (P1), dont l'objectif est de déterminer les dates d'exécution de chaque tâche de manière à minimiser la durée totale d'exécution, tout en respectant les durées et les contraintes de précédence entre les tâches. Il est possible de modéliser ce problème par une modélisation classique, qui sera soit par programmation linéaire, soit par graphe potentiel-tâche.

Pour réaliser la modélisation classique par graphe potentiel-tâche, on applique l'algorithme du plus long chemin sur le graphe correspondant au Tableau 1 et l'on obtient comme date de début : [0 2 2 5 3] et comme date de fin 9.

La modélisation classique par programmation linéaire, quant à elle, retourne comme date de début [0 2 2 5 8] et comme date de fin 9.

On remarque que la date de fin est la même, mais que le temps de la dernière tâche est différent. Ceci est dû aux contraintes de la programmation linéaire, qui, lors de l'optimisation, trouve que 8 est la meilleure valeur, alors que 3 répond aussi aux contraintes.

2 Ordonnancement avec contraintes de précédence et de ressources

2.1 Relaxation des contraintes et des ressources

Les codes de cette partie sont dans le fichier *Jobshop*.

Dans cette partie, une relaxation (R), permettant d'ignorer les contraintes de ressources du job-shop, est utilisée afin de montrer que (R) est équivalent au problème (P1).

Nous pouvons remarquer que (R) peut s'écrire sous la forme :

Opération i	**1**	**2**	**3**	**4**	**5**	**6**
Durée	6	7	0	3	5	1
Condition de début	/	Après fin de 1	Après fin de 2	/	Après fin de 4	Après fin de 5

On constate que (R) s'écrit sous la même forme que (P1), donc (R) est équivalent à (P1).

Nous allons maintenant construire et résoudre le programme linéaire et le graphe potentiel-tâche associés aux données du Tableau 2(a) pour (R).

Nous obtenons ainsi avec le programme linéaire et le graphe potentiel-tâche les dates de début : [0 6 13 0 3 8] avec une date de fin égale à 13.

Nous observons que l'opération 2 pour le travail 1 commence à 6 et que celle correspondant à l'opération 5 pour le travail 2 à 3. À la vue des temps donnés dans le tableau, cela signifierait que les deux opérations ont été réalisées en même temps, ce qui est impossible, car elles utilisent la même machine, qui ne peut exécuter qu'une opération à la fois.

La solution obtenue ne respecte donc pas les contraintes du Tableau 2(b).

2.2 Résolution

La résolution est codée dans le fichier *Jobshop*.

Nous avons d'abord résolu le problème du Tableau 3 avec une PSE basée sur la relaxation linéaire du modèle avec BigM. Pour cela, nous avons repris les codes donnés pour

le Tableau 1 et avons ajouté la variable binaire associée à l'opération 3 du travail 1 et à l'opération 5 du travail 2, ainsi que les conditions associées à cette variable selon la forme donnée dans l'énoncé.

Nous avons ensuite codé la partie sur la PSE basée sur le graphe disjonctif.

Pour réaliser cette partie, il est nécessaire de déterminer si un graphe est sondable ou non. Pour qu'un graphe soit sondable, il faut que :

- pour sa partie conjonctive U : $\forall(ij, ik) \in U$, on ait $t_{ik} - t_{ij} \geq p_{ij}$
- et pour sa partie disjonctive D : $\forall(ij, kl) \in D$, on ait soit $t_{kl} - t_{ij} \geq p_{ij}$, soit $t_{ij} - t_{kl} \geq p_{kl}$

Nous avons donc réalisé la fonction *grapheSondable* qui renvoie si un graphe est sondable.

Puis nous avons réalisé la fonction *PSEGrapheDisjonctif* qui réalise le code de la PSE. Celle-ci repose sur les principes suivants :

- On définit une pile *memoire* avec tous les graphes à analyser et une pile *fils*, permettant d'indiquer quel sous-arbre est analysé, car, dans le modèle du graphe disjonctif, chaque nœud a deux fils ou aucun. Par convention, 1 représente le fils de gauche, quand la variable binaire est supérieure ou égale à 1, et 2 celui de droite, quand cette variable est inférieure ou égale à 0.
- On ajoute une valeur à la pile *fils*, qui n'a pas d'importance, car elle ne sera pas exploitée. Mais elle est nécessaire, car tant que la pile *memoire* est non vide, on dépile *fils*. Or sans cette valeur, impossible de dépiler *fils* à la tout dernière étape.
- On ajoute également le graphe initiale à la pile des graphes.
- Tant qu'il y a des graphes à analyser, on réalise les étapes suivantes :
  - récupération du graphe à analyser : le premier de la pile, auquel on applique notre algorithme du plus long chemin, afin de récupérer la liste des temps et le temps maximal.
  - Si le temps maximal vaut -Inf,
    - alors il n'y a pas de résultat et on affiche TA.
    - Sinon, on regarde si ce graphe est sondable grâce à la fonction *grapheSondable* :
      - Si ce n'est pas le cas, on analyse alors les deux graphes possédant la paire de disjonction k. Pour cela, on ajoute les deux fils du nœud à la pile, en commençant par celui de droite (2), puis celui de gauche (1). On calcule ensuite les deux graphes obtenus à partir du graphe en cours d'analyse, en ajoutant l'arc de la paire de disjonction dans les deux sens. Par exemple, pour la paire de disjonction (2, 4), on ajoute en premier le graphe possédant l'arc de 2 à 4, puis celui possédant l'arc de 4 à 2. Puis, on augmente k pour analyser la prochaine paire de disjonction.
      - Si c'est le cas, alors :
        - si le temps obtenu est plus grand que celui en mémoire, on retourne TO, sinon on retourne TR
        - sinon on met à jour le meilleur temps et la meilleure solution.

        On récupère ensuite le premier sens du tableau. Si celui-ci vaut 2, alors on dépile jusqu'à temps de trouver 1 en décrémentant k. Ainsi, on retourne à la profondeur en attente d'analyse.

Nous avons ainsi montré que les résultats obtenus sont identiques pour les deux tableaux et pour les deux méthodes.

Nous observons, que lors de l'ajout de contraintes ou d'opérations, le nombre de nœuds augmente, de même que les temps de calcul.