Méthodes approchées pour la résolution de problèmes d'ordonnancement

MS ValDom

MJ. HUGUET
homepages.laas.fr/huguet

Objectifs du projet

L'objectif de ce projet est de vous faire concevoir et développer des méthodes approchées pour résoudre un problème d'optimisation combinatoire donné.

Vous avez libre choix du langage de programmation et des outils de développement. Lors des séances de TP, les encadrants sont là pour vous apporter des conseils méthodologiques sur la démarche à mettre en oeuvre pour développer une méta-heuristique.

1 Problème d'ordonnancement étudié : Job Shop

1.1 Définition du problème et modélisation

Le Job Shop est un problème d'ordonnancement dans lequel on dispose :

- d'un ensemble J de n Jobs (ou travaux) décomposé en une succession de tâches (ou activités ou opérations). Pour un job j, on note n_j le nombre de tâches.
- d'un ensemble R de m ressources disjonctives pour réaliser ces tâches.

Toute opération i d'un job j est notée (i, j). Avec $1 \leq i \leq n_j$ et $n_j \leq m$. Au maximum le problème comporte $n * m$ opérations. On note T l'ensemble des opérations.
On dit qu'une ressource est **disjonctive** si elle est disponible en un exemplaire unique et ne pouvant réaliser qu'une seule tâche à la fois. On supposera que les opérations d'un job donné utilise toutes des ressources différentes de R : ainsi pour chaque job il y a au maximum m opérations différentes.
Les contraintes sont les suivantes :

- toute opération (i, j) d'un job j nécessite une unique ressource donnée, notée k, pendant une durée $p_{(i,j)}$,
- chaque ressource k est disponible en un seul exemplaire,
- dans tout job j, l'opération (i, j) est successeur de l'opération $(i-1, j)$, $\forall i \in [2, n_j]$
- les jobs peuvent débuter à la date 0.

L'objectif est de minimiser la durée totale de l'ordonnancement aussi appelée *makespan*. La résolution de ce problème nécessite donc de déterminer la date de début de chaque opération.
Pour modéliser ce problème, on peut utiliser des variables entières représentant les dates de début $st_{(i,j)}$ des différentes opérations (i, j). On introduit dans le modèle deux opérations fictives, notées S et F représentant le début et la fin de l'ordonnancement. Ainsi, S précède chacune des premières opérations des jobs et F succède à chacune des dernières opérations des jobs.
On note T_k l'ensemble des opérations devant être effectuées par une ressource k.
Le modèle s'écrit alors :

$$Min(st_F) \tag{1}$$

$$st_{(i-1,j)} + p_{(i-1,j)} \leq st_{(i,j)},\ \forall j \in J, \forall i \in [2, n_j] \tag{2}$$

$$(st_{(i,j)} + p_{(i,j)} \leq st_{(i',j')}) \ \vee \ (st_{(i',j')} + p_{(i',j')} \leq st_{(i,j)}),\ \forall (i, j), (i', j') \in T_k, \forall k \in R \tag{3}$$

$$st_{(i,j)} \in \mathbb{N}^+,\ \forall (i, j) \in T \tag{4}$$

L'objectif (1) correspond à la minimisation de la date de début de la dernière opération du problème, c'est à dire que l'on cherche à minimiser la date de fin de l'ordonnancement ou sa durée totale. Les contraintes (2) expriment les précédences entre les opérations d'un même job. Les contraintes (3) traduisent les contraintes de partage de ressource. Une ressource étant disponible en quantité unitaire, toute paire de tâche nécessitant une même ressource doit donc être séquencée sur cette ressource. Les contraintes (4) spécifient que les dates de début de chacune des opérations sont positives.

Note : les contraintes (3) peuvent être linéarisées par l'introduction d'une constante ayant une très grande valeur (notée M) et de variables booléennes $x^k_{(i,j)(i',j')}$ représentant le séquencement de deux opérations sur une ressource k. $x^k_{(i,j)(i',j')}$ vaut 1 si (i, j) précède (i', j') et 0 sinon.

$$st_{(i,j)} + p_{(i,j)} + M.(x^k_{(i,j)(i',j')} - 1) \leq st_{(i',j')} \tag{5}$$

$$st_{(i',j')} + p_{(i',j')} - M.x^k_{(i,j)(i',j')} \leq st_{(i,j)} \tag{6}$$

1.2 Illustration

Soit un problème de Job Shop composé de 2 jobs et de 3 ressources. Pour chaque opération de chaque job, le tableau ci-dessous donne la ressource nécessaire et la durée de réalisation sur cette ressource.

J_1	$r_1, 3$	$r_2, 3$	$r_3, 2$
J_2	$r_2, 2$	$r_1, 2$	$r_3, 4$

Les contraintes (2), (3) et (4) de ce problème peuvent être représentées par le graphe de la figure 1 sur lequel on a introduit les opérations fictives S et F. Dans ce graphe, un arc entre deux sommets $st_{(i,j)}$ et $st_{(i',j')}$, représentent la contrainte $st_{(i',j')} \geq st_{(i,j)} + l_{(i,j)}^{(i',j')} \Leftrightarrow st_{i',j')} - st_{(i,j)} \geq l_{(i,j)}^{(i,j)}$ où $l_{(i,j)}^{(i',j')}$ est la valeur de la distance entre $st_{(i,j)}$ et $st_{i',j')}$ (pour le problème étudié, cette distance vaut $p_{(i,j)}$). Les arcs représentés en trait plein représentent les contraintes (2) et (4) : ils traduisent les contraintes temporelles du problèmes et sont appelés arcs conjonctifs. Les arcs représentés en pointillé correspondent aux contraintes (3) : ils traduisent les contraintes de partage de ressources ; ils sont appelés arcs disjonctifs.
Pour obtenir une solution admissible à un problème de Job Shop, il faut orienter chacun des arcs disjonctifs sans créer de cycle de longueur positive dans le graphe.

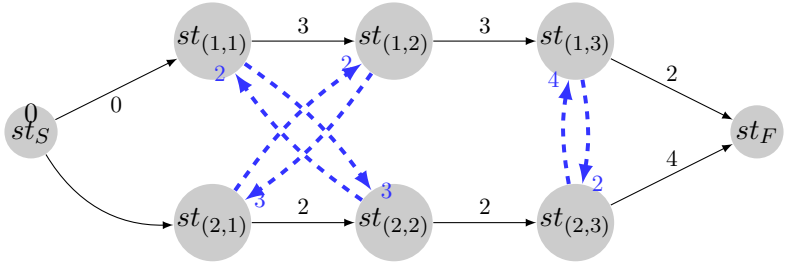

FIGURE 1 – Exemple de Job Shop

Le graphe de la figure 2 (sans cycle de longueur positive) représente une solution pour ce problème.

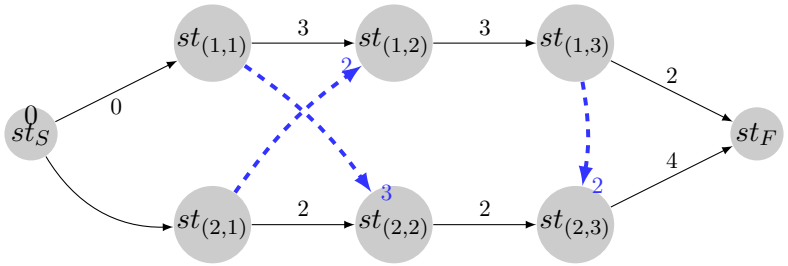

FIGURE 2 – Solution de l'exemple

Question. Quel est le cout de cette solution ? Comment est-il calculé ?
Question. Une solution peut également se représenter sur un diagramme de Gantt. Tracez ce diagramme.

Ce problème "académique" a de nombreuses applications et variantes. Une étude récente[4] en présente une synthèse pour le secteur de l'industrie 4.0.

2 Déroulement du projet

Lire attentivement tout le sujet du projet pour avoir une vue d'ensemble.

2.1 Séance préparatoire :

2.1.1 Analyse du sujet

Cette étape d'analyse doit vous permettre de :

- déterminer la complexité du problème ;
- prendre connaissance du format des instances de Job Shop.
- réfléchir à une représentation des solutions ;
- réflechir à une procédure efficace d'évaluation d'une solution ;

Les instances à traiter sont disponibles sur le site `https://github.com/tamy0612/JSPLIB/tree/master/instances`. Considérez en priorité les instance suivantes :

- `ft06`, `ft10`, `ft20`
- `la01` à `la40`
- `ta01` à `ta50`

Pour ces instances les meilleurs résultats connus sont reportés dans le document `BestKnownResults`.

2.1.2 Finalisation d'une méthode exacte

Terminez une méthode exacte pour résoudre les problèmes de Job Shop (en utilisant un modèle PPC ou un modèle PLNE). Cette étape permettra de comparer les performances relatives de méthodes exactes et approchées.

L'article [3] vous donne des pistes pour un modèle PLNE. Pour un modèle PPC, reprenez les sujets de TP ou regardez les présentations sur ce site : `https://fr.slideshare.net/PhilippeLaborie/solving-industrial-scheduling-problems-with-constraint-programming`.

Documentation DOCplex : `http://ibmdecisionoptimization.github.io/docplex-doc/`

Lancez un script pour évaluer ce ou ces modèles sur diverses instances (vous pouvez utiliser un des serveurs de calcul du GEI pour cela).

2.2 Séance 1 et 2 : Heuristiques gloutonnes

Un schéma général d'heuristique est présenté dans l'article [1] et est résumé ci-dessous :

1. se placer à une date t égale à la plus petite date de début des opérations
2. construire l'ensemble des opérations pouvant être réalisées à la date t
3. sélectionner l'operation (i, j) de plus grande priorité
4. placer (i, k) au plus tôt sur la ressource k qui la réalise (en respectant les contraintes de partage de ressources, c'est à dire en vérifiant la date de disponibilité de la ressource k)
5. recommencer en (3) tant qu'il reste des opérations à ordonnancer
6. recommencer en (2) en incrémentant la date t (à la prochaine date possible compte tenu des dates de début des opérations)

Selon la manière dont sont gérées les priorités entre les opérations on obtient différentes solutions d'ordonnancement. Les règles de piorité classiquement utilisées sont :

- EDD (Earliest Due Date) : donne la priorité à la tâche ayant la plus grande date de fin ;
- SPT (Shortest Processing Time) : donne la priorité à la tâche ayant la plus petite durée ;
- LPT (Longest Processing Time) : donne la priorité à la tâche ayant la plus grande durée

Travail à faire

1. Concevoir et coder une heuristique gloutonne. L'heuristique développée est-elle statique ou dynamique ? Les solutions obtenues sont-elles admissibles ?
2. Définir les métriques d'évaluation de cette méthode ;
3. Evaluer cette heuristique sur les instances fournies et pour les métriques définies précédemment ;
4. Concevoir et coder une heuristique « randomisée »
5. Evaluer cette nouvelle heuristique sur les instances fournies et pour les métriques définies précédemment
6. Comparer les résultats obtenus par ces deux heuristiques par rapport à ceux de la littérature et à la méthode exacte
7. Débuter la rédaction d'un rapport présentant le travail effectué

2.3 Séance 3 : Méthode de descente

Dans cette partie du projet, vous allez développer une méthode de descente afin d'améliorer les résultats de l'heuristique gloutonne. Le principe d'une méthode de descente a été présenté dans le cours.

Travail à faire

1. Sélectionner un voisinage ;
2. Définir les métriques d'évaluation de votre méthode ;
3. Evaluer cette méthode sur les instances fournies et pour les métriques définies précédemment ;
4. Comparer les résultats obtenus par rapport aux résultats antérieurs.
5. Débuter la rédaction d'un rapport présentant le travail effectué

2.4 Séance 4 et 5 : Métaheuristique

Vous avez le choix entre deux métaheuristiques : une recherche locale (exploration de voisinages : recuit, tabou, VNS, ...) ou un algorithme génétique. Les articles [2] et [4] présentent une liste de méthodes de la littérature qui peuvent vous inspirer. Par ailleurs le document `Hertz-MetaHeuristiques` fournit de précieux conseils pour concevoir une métaheuristique.

Quelque soit la méthode retenue, vos devez expliquer les choix spécifiques à chacune et proposer une évaluation expérimentale.

3 Rendu de projet

A la fin des séances, vous devrez rendre un programme exécutable et un rapport.

3.1 Contraintes de développement

Choisissez un langage standard (Python, Java, ...) et créez un dépot git pour votre projet. Vos codes doivent pouvoir s'exécuter dans un environnement Linux tel que celui proposé dans les salles de TP. Supposons que votre exécutable soit placé dans le repertoire `FJS/`, les exemples seront dans un répertoire appelé `FJS/instances/` et les résultats devront se trouver dans un répertoire appelé `FJS/results/`.
Vos programmes doivent permettre :

- de **résoudre une instance** donnée à partir de son nom et de générer un fichier de résultat ;
- de **résoudre un ensemble d'instances** et de générer un fichier de résultat ;
- de **vérifier** si une solution est correcte (à partir du nom d'une instance).

Le contenu des fichiers de résultat pour un ensemble d'instances doit être :

- le nom de l'ensemble d'instance et le nombre d'instances de cet ensemble
- Pour chaque instance : son nom
- puis la valeur du makespan, le temps de calcul et les informations spécifiques à la métaheuristique
- Les valeurs moyennes pour le makespan, le temps de calcul et les informations spécifiques

3.2 Contenu du rapport

Après la dernière séance, vous devez remettre un rapport(maximum 15 pages) décrivant

1. les structures choisies pour représenter une solution
2. la méthode d'évaluation de solutions
3. l'heuristique gloutonne et « randomisée »développée
4. le(s) voisinage(s) utilisé(s) pour la méthode de descente
5. les choix effectués pour la métaheuristique développée
6. les résultats obtenus par ces différentes méthodes sur les instances proposées
7. une comparaison de vos résultats avec ceux de la littérature
8. n'oubliez pas l'introduction ni la conclusion.

Pour ce rapport, il est inutile de rappeler la description d'un problème de Job Shop.

Références

[1] Jacek Blazewicz, Wolfgang Domchke, and Erwin Pesch. The job shop scheduling problem : Conventional and new solution techniques. *European Journal of Operational Research*, 93(1) :1–33, Aug 1996.

[2] Banu Çaliş and Serol Bulkan. A research survey : review of ai solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5) :961–973, Oct 2015.

[3] Wen-Yang Ku and J. Beck. Mixed integer programming models for job shop scheduling : A computational analysis. *Computers & Operations Research*, 73, 04 2016.

[4] Jian Zhang, Guofu Ding, Yisheng Zou, Shengfeng Qin, and Jianlin Fu. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30(4) :1809–1830, Apr 2019.