

Méthodes Approchées pour la Résolution de Problèmes d'Ordonnancement

Partie 1

Arthur Bit-Monnot, Marie-Jo Huguet

1 Étapes de mise en place

1.1 Discord

- rejoignez le serveur Discord pour ces TP : <https://discord.gg/KyUbCCT>.
- indiquez votre *Prénom Nom* comme pseudo, pour que l'on puisse vous identifier
- en cas de problème technique, vous pourrez vous adresser au chan *support-technique* de ce serveur.

1.2 Document de suivi

- inscrivez vous dans le document de suivi : <https://docs.google.com/spreadsheets/d/1QAZIWaTCvrMILLuwuVFmCD8Plr9QvOoPVpIR4g1PSBk/edit?usp=sharing>
- à chaque étape franchie dans le TP, ajoutez un X dans la case correspondante

1.3 Récupération du code

- Récupérez la base de code sur Github : <https://github.com/insa-4ir-meta-heuristiques/template-jobshop>
- Suivez les instructions dans le README pour vous assurer que le code compile.
- Importez le projet dans votre IDE préféré. Tous doivent pouvoir supporter l'import de projet gradle et quelques liens sont donnés en bas du README.

2 Prise en main

2.1 Représentation d'un problème de JobShop

Vous trouverez dans le dossier `instances/` un ensemble d'instances communément utilisées pour le problème de jobshop. Si l'on considère l'instance vu dans les exercices, constituée de deux jobs avec trois tâches chacun :

J_1	$r_{1,3}$	$r_{2,3}$	$r_{3,2}$
J_2	$r_{2,2}$	$r_{1,2}$	$r_{3,4}$

L'instance est nommée `aaa1` décrite ci-dessus est donnée dans le fichier `instances/aaa1` :

```
# Fichier instances/aaa1
2 3 # 2 jobs and 3 tasks per job
0 3 1 3 2 2 # Job 1 : (machine duration) for each task
```

```
1 2 0 2 2 4 # Job 2 : (machine duration) for each task
```

La première ligne donne le nombre de jobs et le nombre de tâches par jobs. Le nombre de machine est égal au nombre de tâches. Chacune des lignes suivantes spécifie la machine (ici numérotées 0, 1 et 2) et la durée de chaque tâche. Par exemple la ligne 0 3 1 3 2 2 spécifie que pour le premier job :

- 0 3: la première tâche s'exécute sur la machine 0 et dure 3 unités de temps
- 1 3: la deuxième tâche s'exécute sur la machine 1 et dure 3 unités de temps
- 2 2: la troisième tâche s'exécute sur la machine 2 et dure 2 unités de temps

2.2 Base de code

Il vous est fourni une base de code pour faciliter votre prise en main du problème. Vous trouverez dans la classe `jobshop.Main` un point d'entrée pour réaliser des tests de performance de méthodes heuristiques. Les autres points d'entrée du programme sont les tests déjà présent ou pouvant être ajoutés dans le dossier `src/test/java/`.

2.2.1 Problème et Solution

- classe `jobshop.Instance` qui contient la représentation d'un problème et une méthode pour parser un fichier de problème.
- classe `jobshop.Schedule` qui contient la représentation directe, associant à chaque tâche une date de début. La classe `schedule` sert notamment à la représentation d'une solution et toute autre représentation doit pouvoir être traduite dans un `Schedule`

2.2.2 Représentation

- une classe abstraite `jobshop.Encoding` dont les sous classes sont des représentations d'une solution au JobShop.
- une classe `jobshop.encoding.NumJobs` qui contient une implémentation de la représentation par numéro de job

2.2.3 Solveurs

Deux solveurs très simples basés sur une représentation par numéro de job. Les nouveaux solveurs doivent être ajoutés à la structure `jobshop.Main.solvers` pour être accessibles dans le programme principal.

- `basic` : méthode pour la génération d'une solution pour une représentation par numéro de job
- `random` : méthode de génération de solutions aléatoires par numéro de job

2.3 À faire : manipulation de représentations

Ouvrez la méthode `DebuggingMain.main()`.

- Pour la solutions en représentation par numéro de job donnée, calculez (à la main) les dates de début de chaque tâche
- implémentez la méthode `toString()` de la classe `Schedule` pour afficher les dates de début de chaque tâche dans un `schedule`.
- Vérifiez que ceci correspond bien aux calculs que vous aviez fait à la main.

Création d'une nouvelle représentation par ordre de passage sur les ressources :

- Créer une classe `jobshop.encodings.ResourceOrder` qui contient la représentation par ordre de passage sur ressources vue dans les exercices (section 3.2). Il s'agit ici d'une représentation sous forme de matrice où chaque ligne correspond à une machine, et sur cette ligne se trouvent les

tâches qui s'exécutent dessus dans leur ordre de passage. Pour la représentation d'une tâche dans la matrice, vous pouvez utiliser la classe `jobshop.encodings.Task` qui vous est fournie.

- Pour cette classe, implémentez la méthode `toSchedule()` qui permet d'extraire une représentation directe. Pour l'implémentation de cette méthode `toSchedule()`, il vous faut construire un `schedule` qui associe à chaque tâche une date de début (vous pouvez regarder l'implémentation pour `JobNums` pour en comprendre le principe). Pour construire ce `schedule` il faudra que vous mainteniez une liste des tâches qui ont été schedulées. Cette liste est initialement vide. À chaque itération de l'algorithme, on identifie les tâches executables. Une tâche est executable si
 - son prédécesseur sur le job a été schedulé (si c'est la tâche (1, 3), il faut que les tâches (1,1) et (1,2) aient été schedulées)
 - son prédécesseur sur la ressource a été schedulé (l'ordre sur passage sur les ressources est précisément ce qui vous est donné par cette représentation).
- Ajouter des tests dans `src/test/java/jobshop` permettant de vérifier que vos méthodes fonctionnent bien pour les exemples traités en cours (instance `aaa1`). Vous pouvez pour cela vous inspirer et ajouter des cas de test à `EncodingTests`.

Changement de représentation :

- pour les deux représentations `ResourceOrder` et `JobNums`, créez des méthodes permettant de créer cette représentation depuis un `Schedule`.
- utilisez la pour tester la conversion de `ResourceOrder` vers `JobNums` et vice-versa.

3 Heuristiques gloutonnes

On souhaite maintenant implémenter une méthode gloutonne pour calculer une solution. Cette méthode gloutonne doit être basée sur la représentation par ordre de passage. On rappelle ci-dessous le grand principe d'une méthode gloutonne pour le jobshop :

- **Initialisation** : Déterminer l'ensemble des tâches réalisables (initialement, les premières tâches de tous les jobs)
- **Boucle** : tant qu'il y a des tâches réalisables
 - Choisir une tâche dans cet ensemble et placer cette tâche sur la ressource qu'elle demande (à la première place libre dans la représentation par ordre de passage)
 - Mettre à jour l'ensemble des tâches réalisables

On dit qu'une tâche est *réalisable* si tous ses prédécesseurs ont été traités (précédences liées aux jobs). Il faut noter que chaque tâche doit apparaître exactement une fois dans l'ensemble des tâches réalisables et que toutes les tâches doivent être traitées.

Un paramètre déterminant dans la construction d'une heuristique gloutonne est la manière dont est choisie la prochaine tâche à exécuter parmi toutes les tâches réalisables. Cette décision est prise à partir de règles qui permettent de donner une priorité plus élevée à certaines tâches. Pour le JobShop, on peut notamment considérer les règles de priorité suivantes :

- SPT (Shortest Processing Time) : donne priorité à la tâche la plus courte ;
- LPT (Longest Processing Time) : donne priorité à la tâche la plus longue ;
- SRPT (Shortest Remaining Processing Time) : donne la priorité à la tâche appartenant au job ayant la plus petite durée restante;
- LRPT (Longest Remaining Processing Time) : donne la priorité à la tâche appartenant au job ayant la plus grande durée

3.1 À faire

- Créer un nouveau solveur implémentant une recherche gloutonne pour les priorités SPT, LRPT, basée sur la représentation par ordre de passage sur les ressources
- Évaluer ces heuristiques sur les instances fournies et pour la métrique d'écart fournie. Pour les tests de performance, on privilégiera les instances **ft06**, **ft10**, **ft20** et les instances de Lawrence **1a01** à **1a40**.

On cherche maintenant à améliorer ces méthodes gloutonnes en limitant le choix de la prochaine tâche à celles pouvant commencer au plus tôt. Par exemple, si il y a trois tâches réalisables $(1, 2)$, $(2, 1)$ et $(3, 3)$ dont la date de début au plus tôt sont respectivement 4, 6 et 4 ; alors le choix de la prochaine tâche à faire sera réalisée entre les tâches $(1, 2)$ et $(3, 3)$ qui sont les deux pouvant commencer au plus tôt (c'est à dire à la date 4). Pour départager ces deux tâches, on utilisera l'une des règles de priorités (par exemple LRPT).

- Pour les deux priorités SPT et LRPT, implémenter la restriction aux tâches pouvant commencer au plus tôt. Ces nouvelles règles sont appelées **EST_SPT** et **EST_LRPT**.