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/1QAZlWaTCvrMlLLuwuVFmCD8Plr9QvOoPVpIR4g1PSBk/edit?usp=sharing
- à chaque étape franchie dans le TP, ajountez un `X` dans la case correspondante

1.3 Récuperation 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 donnée 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'execute sur la machine `0` et dure `3` unités de temps
- `1 3`: la deuxième tâche s'execute sur la machine `1` et dure `3` unités de temps
- `2 2`: la troisième tâche s'execute 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 representation 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 reprsentation 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 representation 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 regardez 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 schédulé. Cette liste est initialement vide. À chaque itération de l'algorithme, on identifie les tâches executables. Une tâche est executable si
  - son prédecesseur sur le job a été schedulé (si c'est la tache (1, 3), il faut que les tâches (1,1) et (1,2) aient été schédulée)
  - son prédecesseur sur la ressource a été schedulé (l'ordre sur passage sur les ressources est précisement 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 gloutonne

Un schéma général d'heuristique est présenté dans l'article (Blazewicz, Domchke, and Pesch 1996) 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) réalisable de plus grande priorité
4. placer (i, j) 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 :

- 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

3.1 À faire

- Créer un nouveau solveur implémentant une recherche gloutonne pour les priorités SPT et LPT, basé sur la représentation par ordre de passage sur les ressources
- Evaluer ces heuristiques sur les instances fournies et pour la métrique d'écart fournie.
- (optionnel) Concevoir une version randomisée de ces heuristiques où une partie des choix est soumise à un tirage aléatoire
- Débuter la rédaction d'un rapport présentant le travail effectué

Pour les tests de performance, on privilégiera les instance `ft06`, `ft10`, `ft20` et les instances de Lawrence `la01` à `la40`.

4 Recherche exhaustive

Dans le but de garantir l'optimalité des solutions trouvée, implémentez une recherche exhaustive basée sur une recherche en profondeur d'abord (Depth-First Search).
On notera que par rapport à la méthode gloutonne, une recherche exhaustive doit permettre de tester l'ensemble des opérations pouvant être réalisée à instant t (étape (2) de la méthode gloutonne ci-dessus).
On ne cherchera pas à optimiser cette méthode, des techniques plus adaptées à la recherche exhaustive pour de tels problèmes seront traitées en 5ème année.

Quelle semble être la taille limite d'une instance pour permettre une recherche exhaustive ?

5 Références

Blazewicz, Jacek, Wolfgang Domchke, and Erwin Pesch. 1996. "The Job Shop Scheduling Problem: Conventional and New Solution Techniques." *European Journal of Operational Research* 93 (1): 1–33.