Devoir 3 - Optimisation de la production de meubles

Remise le 18/04/2023 **(avant minuit) sur Moodle pour tous les groupes.**

Consignes

- Le devoir doit être fait par groupe de 2 au maximum. Il est fortement recommandé d'être 2.
- Lors de la soumission sur Moodle, donnez votre rapport en **PDF** ainsi que votre code **commenté** à la racine d'un seul dossier compressé nommé (`matricule1_matricule2_Devoir3.zip`).
- Indiquez vos noms et matricules en commentaires au dessus des fichiers `.py` soumis.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale pour le devoir.

Énoncé du devoir

L'atelier *LaTulipe* est un manufacturier de meubles faits sur mesure. L'atelier fournit en meubles de grands clients institutionnels comme des hôpitaux ou des universités. Pour un client donné, l'atelier reçoit une commande de meubles. La construction d'un meuble se divise en plusieurs *tâches*. Les tâches sont définies par des relations de *précédence* (c'est-à-dire qu'une tâche ne peut commencer que si certaines autres tâches sont déjà finies), des *durées* et nécessitent une certaine quantité de *ressources*. Les relations de précédence expriment les tâches qui ne peuvent commencer avant qu'une tâche en question ne soit terminée. Par exemple, on ne peut pas vernir une table avant qu'elle n'ait été sablée. Si une tâche n'a pas de précédence, elle peut être entamée sans contrainte sur son début d'exécution. Finalement, une tâche peut avoir plusieurs précédences.

Toutes mes ressources considérées dans ce projet sont *renouvelables*. C'est-à-dire que la quantité disponible de la ressource dépend seulement de son utilisation actuelle. Par exemple, si l'on considère la tâche de couper des morceaux de bois avec une scie, et qu'on a 8 scies disponibles, la ressource de la scie est une ressource renouvelable de capacité 8. Autrement dit, après avoir été utilisée par une activité, elle retrouve sa capacité d'origine. En pratique, on pourrait également avoir des ressources non-renouvelables, qui, une fois utilisée, ne sont pas récupérée (par exemple, une quantité de bois pour la fabrication). L'objectif est de minimiser le temps total nécessaire pour compléter toutes les tâches (le *makespan*).

Formalisation du problème

On considère un ensemble de n tâches, chacune ayant un temps de traitement p_i, un ensemble de m ressources renouvelables, chacune ayant une capacité c_r, et un horizon temporel H. On pose $r_{i,j,t}$ comme étant la consommation de la ressource renouvelable j de la tâche i au temps t. Cette consommation est nulle si la tâche n'est pas exécutée au temps t et a une valeur prédéfinie correspondant à l'exigence de la tâche pour cette ressource, si elle est exécutée. De plus, il peut y avoir des relations de précédence entre les tâches, de sorte que la tâche i doit être terminée avant que la tâche j puisse commencer. Nous désignons une telle relation de précédence par $(i \rightarrow j)$.

Modèle mathématique simplifié

Soit x_i la date de début de la tâche $i \in \{1,\dots,n\}$. Le modèle suivant vise à minimiser le *makespan* tout en respectant les contraintes de précédences (Contrainte (2)) et de capacité des ressources (Contrainte (3)). Notez bien que cette dernière contrainte vérifie qu'à chaque instant, la consommation de chaque ressource n'est pas excédée. Finalement, la Contrainte 4 indique qu'aucune activité peut démarrer avant le temps zéro (début de la planification).

$$\min_{x} \max_{i \in 1,\dots n} (x_i + p_i) \tag{1}$$

$$\text{subject to } x_i + p_i \leq x_j \qquad \text{pour toutes les précédences } (i \rightarrow j) \tag{2}$$

$$\sum_{i=1}^{n} r_{i,j,t} \leq c_r \qquad \forall j \in \{1,\dots,m\} \wedge \forall t \in \{1,\dots,H\} \tag{3}$$

$$x_i \geq 0 \qquad \forall i \in \{1,\dots,n\} \tag{4}$$

Description des instances

Différentes instances vous sont fournies. Elles sont nommées selon le schéma `instance_X_N.txt` avec X le nom de l'instance, et N le nombre de tâches dans le problème (excluant les noeuds triviaux de départ et de fin). Chaque fichier d'instance provient de la librairie PSPLIB [1] et a le format suivant [2].

```
************************************************************************
#jobs (incl. supersource/sink ): 32
horizon                       : 181
#resources                     : 4
************************************************************************
PRECEDENCE RELATIONS:
jobnr.  #successors   successors
   1          3           2   3   4
   2          3           7   8  18
  ...
************************************************************************
REQUESTS AND DURATION
jobnr. duration  R 1  R 2  R 3  R 4
------------------------------------------------------------------------
  1       0       0    0    0    0
  2       3       0    0    7    0
  3       8       2    0    0    0
  4       5       0    0    0    9
  ...
************************************************************************
RESOURCE AVAILABILITIES:
  R 1  R 2  R 3  R 4
   13    8   11   19
************************************************************************
```

Les informations contenues sont les suivantes :

— `#jobs` : le nombre d'activités (*jobs*) de l'instance.

1. `https://www.om-db.wi.tum.de/psplib/`
2. Le format a été légèrement simplifié par rapport à ceux de la librairie.

- `horizon` : l'horizon temporel endéans lequel toutes les activités doivent être exécutées.
- `#resources` : le nombre de ressources de l'instance.
- `precedence relations` : une liste de toutes les précédences. Pour chaque activité (`jobnr`), le nombre de tâches successeurs (`#successors`), ainsi que l'identifiant de chaque tâche successeur (`successors`) sont indiqués.
- `requests and duration` : pour chaque activité, la durée de traitement, et les exigences pour chaque ressource. Notez la présence de 2 tâches triviales, la première et dernière tâche, qui ont une durée nulle et aucune consommation de ressources.
- `resource availabilities` : la capacité maximale pour chaque ressource.

Le format attendu d'une solution est un fichier de n lignes. Chaque ligne contient comme premier élément le numéro d'une tâche et comme deuxième élément le moment où cette tâche commence. **Les fichiers de solution doivent être nommés selon la convention suivante** : solution_{instance}.txt. A titre d'exemple, l'instance suivante (`instance_A_4_30.txt`) contient 30 tâches et une solution (non faisable) est représentée ci-dessous et doit avoir le nom solution_instance_A_4_30.txt.

```
1,0
2,0
3,8
4,12
5,18
6,21
7,29
8,34
9,43
10,45
11,52
12,61
13,63
14,69
15,72
16,81
17,91
18,97
19,102
20,105
21,112
22,114
23,121
24,123
25,126
26,129
27,136
28,144
29,147
30,154
31,156
32,158
```

Pour faciliter la lecture d'une solution, un outil de visualisation vous est fourni. Celui-ci génère deux image : un graphique de gantt représentant les relations de précédences dans la solution et une image indiquant la quantité de chaque ressource consommée à chaque instant.

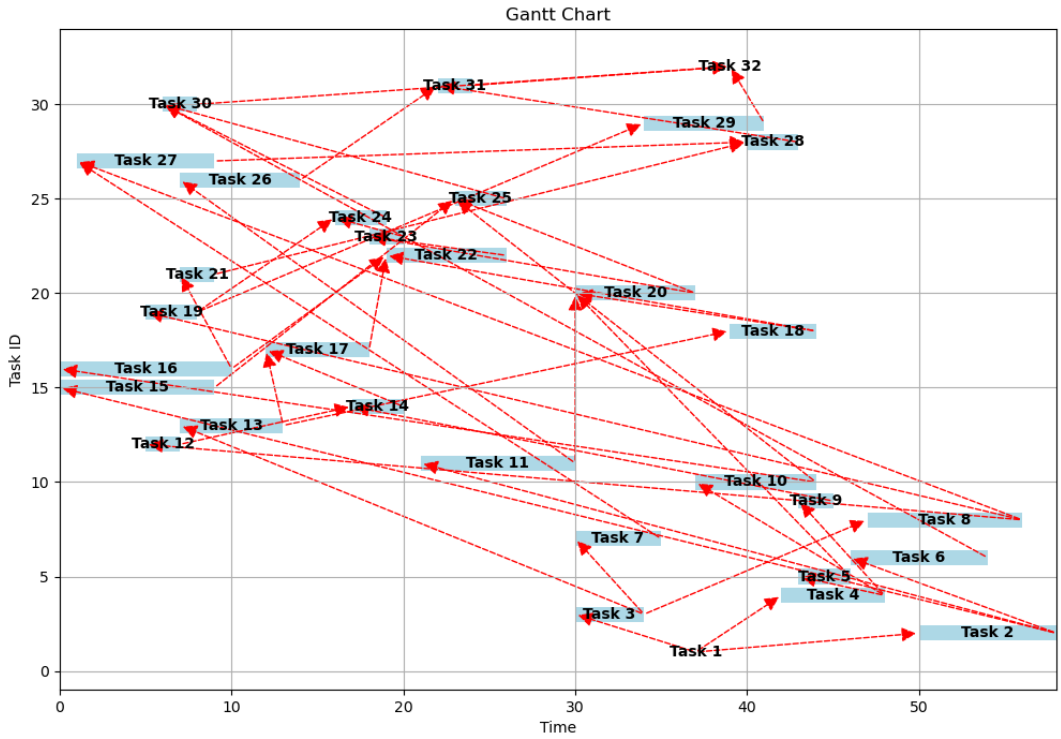

FIGURE 1 – Dépendances entre les tâches.

Implémentation

Vous avez à votre disposition un projet python. Quatre fichiers vous sont fournis.

- `rcpsp.py` qui implémente la classe `RCPSP` pour lire les instances, construire et stocker vos solutions.
- `main.py` vous permettant d'exécuter votre code sur une instance donnée. Ce programme stocke également votre meilleure solution dans un fichier au format texte et sous la forme d'une image.
- `solver_naive.py` qui implémente une résolution triviale du problème.
- `solver_advanced.py` qui implémente votre méthode de résolution du problème.

Vous êtes également libres de rajouter d'autres fichiers au besoin. De plus, 5 instances sont mises à votre disposition.

- `instance_A_30.txt` ; d'optimum connu égal à 51.
- `instance_B_30.txt` ; d'optimum connu égal à 43.
- `instance_C_60.txt` ; d'optimum connu égal à 74.
- `instance_D_60.txt` ; d'optimum connu égal à 114.
- `instance_E_90.txt` ; d'optimum connu égal à 78.

Votre code sera également évalué sur une instance cachée (`secret_X.txt`), de taille semblable à la dernière. Pour vérifier que tout fonctionne bien, vous pouvez exécuter le solveur aléatoire comme suit.

```
python3 main.py --agent=naive --infile=instances/A_30.txt
```

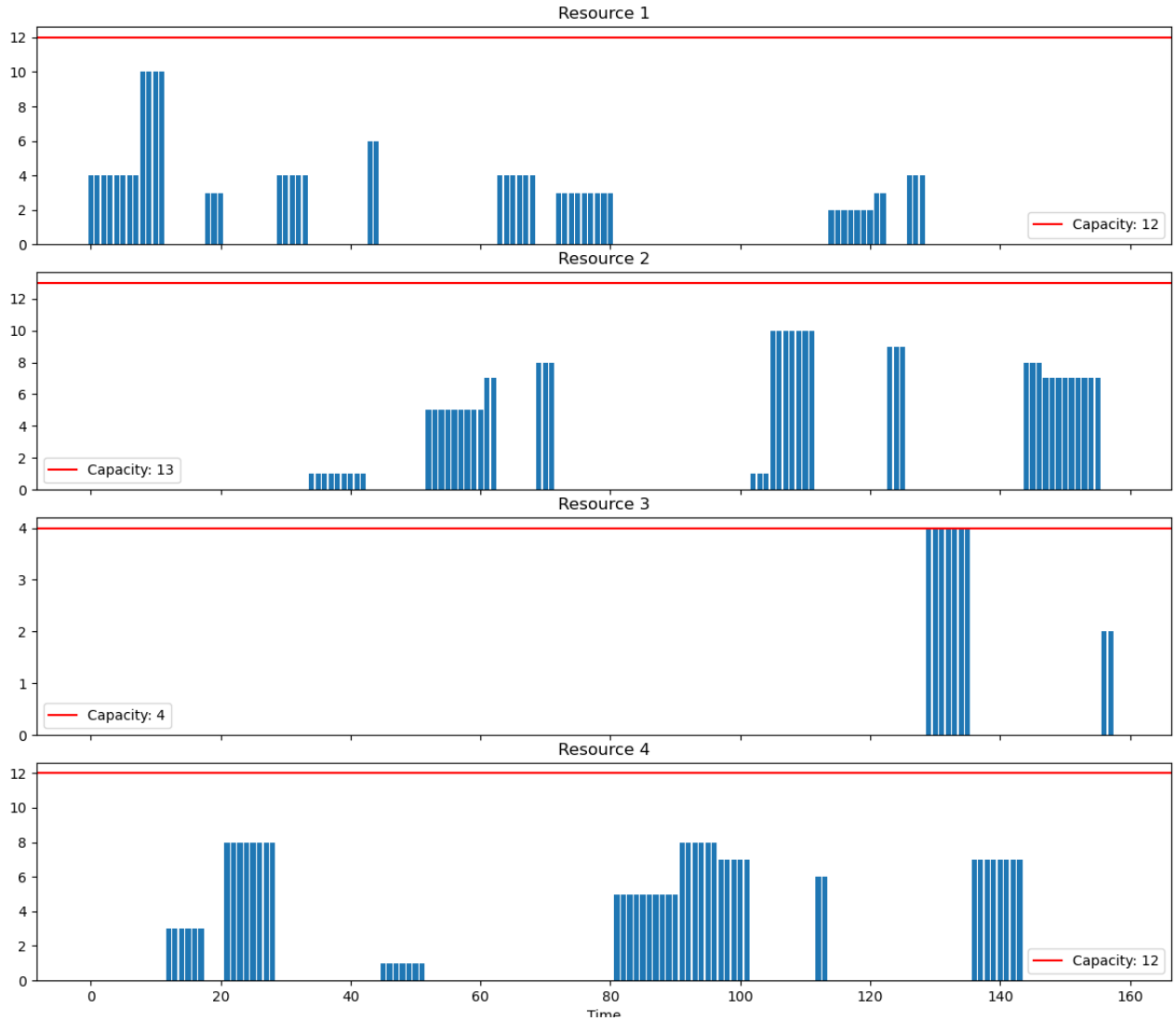

FIGURE 2 – Ressources consommées.

Production à réaliser

Vous devez compléter le fichier `solver_advanced.py` avec votre méthode de résolution. Au minimum, votre solveur doit contenir un algorithme de recherche locale amélioré par au moins une des métaheuristiques suivantes : *recherche locale itérée, recherche à voisinage variable, recherche à voisinage large (adaptatif)*. Vous êtes libres de choisir la métaheuristique que vous souhaitez implémenter.

Comme pour le premier devoir, réfléchissez bien à la définition de votre espace de recherche, de votre voisinage, de la fonction de sélection et d'évaluation. Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur, par exemple en combinant votre approche avec une autre métaheuristique ou de réutiliser des idées des autres devoirs. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=instances/A_30.txt
```

Un rapport succinct (2 pages de contenu, sans compter la page de garde, figures, et références) doit également être fourni. Dans ce dernier, vous devez présenter votre algorithme de résolution, vos choix de conceptions, et vos analyses de complexité. Reportez également les résultats obtenus pour les différentes instances.

Critères d'évaluation

L'évaluation portera sur la qualité du rapport et du code fournis, ainsi que sur les performances de votre solveur sur les différentes instances. Concrètement, la répartition des points (sur 20) est la suivante :

— 10 points sur 20 sont attribués à l'évaluation de votre solveur. Les instances A et B rapportent 1 point si l'optimum est trouvé, et 0 sinon. L'instance C rapporte 2 points si l'optimum est trouvé et diminue progressivement jusqu'à 0 en fonction de la qualité de la solution. Les instances D, E rapportent 2 points si l'optimum est obtenu et diminue progressivement jusqu'à 0 en fonction de la qualité de la solution. Si aucun groupe ne trouve l'optimum pour ces instances, la solution du meilleur groupe est prise comme référence des 2 points. L'instance cachée (X) rapporte entre 0 et 2 points en fonction d'un seuil raisonnable défini par le chargé de laboratoire. Le temps d'exécution est de 20 minutes par instance. Finalement, 1 point est consacré à l'appréciation générale de votre implémentation (bonne construction, commentaires, etc).
— 10 points sur 20 sont attribués pour le rapport. Pour ce dernier, les critères sont la qualité générale, la qualité des explications, et le détail des choix de conception.
— 2 points bonus seront attribués au groupe ayant le meilleur résultat pour l'instance cachée (X).

⚠Il est attendu que vos algorithmes retournent une solution et un coût correct. Un algorithme retournant une solution non cohérente est susceptible de ne recevoir aucun point.

Conseils

Voici quelques conseils pour le mener le devoir à bien :

1. Tirez le meilleur parti des séances de laboratoire encadrées afin de demander des conseils.
2. Inspirez vous des techniques vues au cours, et ajoutez-y vos propres idées.
3. Tenez compte du fait que l'exécution de vos algorithmes peut demander un temps considérable. Dès lors, ne vous prenez pas à la dernière minute pour réaliser vos expériences.
4. Bien que court dans l'absolu, prenez garde au temps d'exécution. Exécuter votre algorithme sur les 4 instances données prend 1h20. Organisez au mieux votre temps de développement et d'évaluation.

Remise

Vous remettrez sur Moodle une archive zip nommée `matricule1_matricule2_Devoir3` contenant :
— Votre code commenté au complet.
— Vos solutions aux différentes instances nommées `solutionX` où X est le nom de l'instance.
— Votre rapport de présentation de votre méthode et de vos résultats, d'au maximum 2 pages.