Devoir 3 - Ordonnancement de la construction de voitures

Remise le 17/04/2022 **(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.
- Soumettez votre rapport en format **PDF** ainsi que vos fichiers de code **commentés** 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.

Enoncé du devoir

Dans ce devoir, il vous est demandé d'ordonnancer la production d'une série de voitures dans une usine. De manière simplifiée, la construction d'une voiture (une *tâche*) est décomposée en une série d'étapes qui correspondent à la réalisation d'une partie de la voiture (construction de la carrosserie, l'intégration du moteur, la pose des roues, assemblage final, etc.) Une étape de construction au sein d'une tâche spécifique est appelée une *opération*. De plus, chaque opération ne peut être exécutée que dans certaines stations, aussi appelées *machines*. Chaque station a la capacité de réaliser un sous-ensemble d'opérations. Ainsi, il n'est pas possible de construire une carrosserie dans une station dédiée à la pose des roues. Par ailleurs, la réalisation de chaque opération a un temps d'exécution qui dépend de la machine sur laquelle l'opération est effectuée.

Etant donné une série de tâches à réaliser, votre mandat est de définir pour chaque opération : (1) la machine sur laquelle elle sera exécutée, et (2) son heure de début d'exécution. L'objectif est de minimiser la durée totale de production (c-à-d, le *makespan*). Deux contraintes importantes doivent être considérées : (1) chaque machine ne peut réaliser qu'une seule opération à la fois, et (2) les opérations d'une tâche doivent s'exécuter dans l'ordre (c-à-d, que l'opération 2 d'une tâche ne peut être exécutée qu'une fois l'opération 1 achevée).

Formellement, le problème d'ordonnancement est défini comme suit. Soit $J = \{j_1, j_2, \ldots, j_n\}$ l'ensemble des *tâches*, soit l'ensemble $O_j = \{o_{j,1}, o_{j,2}, \ldots, o_{j,k}\}$ des *opérations* nécessaires à la réalisation de la tâche j, et soit $M = \{m_1, m_2, \ldots, m_l\}$ l'ensemble des *machines*. Chaque opération o ne peut être exécutée que sur un sous-ensemble $M_o \subseteq M$ de machines. Finalement, on pose $p_{j,o,m}$ comme étant le temps de production d'une opération o de la tâche j sur la machine m. L'objectif est d'ordonnancer toutes les opérations sur les machines dans l'ordre qui minimisera la date de fin de traitement de l'ensemble des jobs. Concernant les contraintes, une machine ne peut réaliser qu'une seule opération à la fois. De plus, pour réaliser l'opération $o_{j,k}$ de la tâche j, il faut que toutes les opérations $o_{j,k'}$ telles que $k' < k$ aient été achevées.

Différentes instances vous sont fournies. Elles sont nommées selon le schéma `factory_X_J_M.txt` avec X le nom de l'instance, J le nombre de tâches à considérer, et M le nombre de machines disponibles. Chaque fichier d'instance contient $N+1$ lignes. Un exemple de fichier vous est proposé ci dessous.

```
2 3
2 [2 1 2 2 4] [2 1 5 3 7]
2 [1 1 4] [2 2 6 3 1]
```

La première ligne indique le nombre d'activités (2) et de machines (3). Chacune des lignes suivantes est associée à une tâche j. La première valeur k d'une ligne indique le nombre d'opérations de la tâche (2 pour la première tâche). Ensuite, une ligne se divise en k blocs d'information, chacun correspondant aux informations propres à une opération. Chaque bloc (délimité par les crochets) d'une ligne se lit comme suit. La première valeur du bloc (2) indique le nombre de machines capables de réaliser l'opération o_k. Les informations du bloc donnent deux informations pour chacune de ces machines : l'identifiant de la machine, et le temps d'exécution de la tâche sur cette machine. Chaque bloc de la ligne est structuré de la même façon. Il en va de même pour chaque autre ligne du fichier. A titre d'exemple, l'instance précédente contient les informations suivantes :

— *Ligne 1* : l'instance a 2 activités et 3 machines.
— *Ligne 2* : la première tâche comporte deux opérations qui ont les caractéristiques suivantes.
  1. La première opération (bloc [2 1 2 2 4]) peut être exécutée sur deux machines. Sur la machine 1, le temps d'exécution est de 2 heures. Sur la machine 2, le temps d'exécution est de 4 heures.
  2. La deuxième opération (bloc [2 1 5 3 7]) peut être exécutée sur deux machines. Sur la machine 1, le temps d'exécution est de 5 heures. Sur la machine 3, le temps d'exécution est de 7 heures.
— *Ligne 3* : la deuxième tâche comporte deux opérations qui ont les caractéristiques suivantes.
  1. La première opération (bloc [1 1 4]) peut être exécutée sur une machine. Sur la machine 1, le temps d'exécution est de 4 heures.
  2. La deuxième opération (bloc [2 2 6 3 1]) peut être exécutée sur deux machines. Sur la machine 2, le temps d'exécution est de 6 heures. Sur la machine 3, le temps d'exécution est de 1 heure.

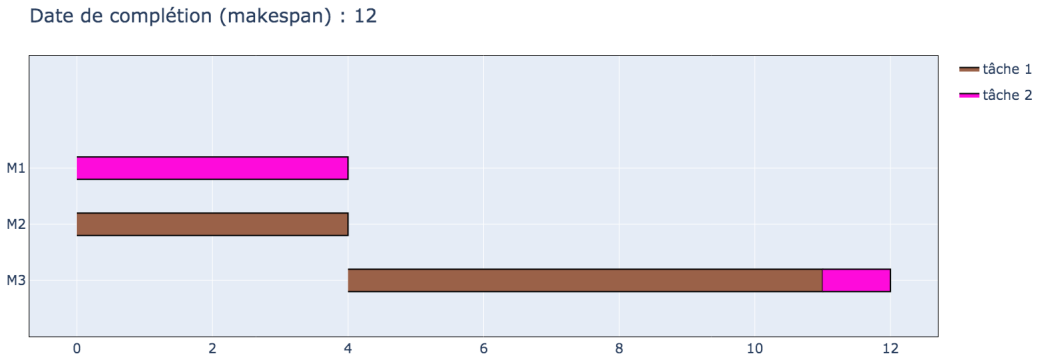
Finalement, notez que les crochets ([ ]) ne sont pas présents dans le fichier d'instance, et ont seulement été utilisés pour faciliter la compréhension du fichier. Le format attendu d'une solution est un fichier de $m+2$ lignes qui décrit les opérations ayant lieu sur chaque machine. Un exemple de solution pour l'instance précédente est présenté ci-dessous.

```
2 3
12

(2,1,0,4)
(1,1,0,4)
(1,2,4,11) (2,2,11,12)
```

La première ligne reprend le nombre d'activités (2) et de machines (3). La deuxième ligne indique le coût de la solution (le *makespan*). Ensuite, chaque autre ligne indique les opérations ayant lieu sur une machine. Chaque tuple (j, o, s, e) reprend les éléments suivants : le temps de début s et de fin e de l'opération o de la tâche j. Ainsi, sur la machine 1, on peut voir que l'opération 1 de la tâche 2 est exécutée du temps 0 jusqu'au temps 4. Comme pour les autres devoirs, une procédure pour générer ce fichier est disponible. Finalement, pour faciliter l'analyse d'une solution, un outil de visualisation vous est fourni.

⚠**Soyez bien sûrs de bien comprendre ce format avant de commencer votre implémentation**



Implémentation

Vous avez à votre disposition un projet python :

- `factory.py` qui implémente la classe `Factory` pour lire les instances, construire, vérifier et stocker vos solutions.
- `main.py` qui vous permet 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 aléatoire du problème.
- `solver_advanced.py` qui implémente votre méthode de résolution du problème.

Vous êtes également lires de rajouter d'autres fichiers au besoin. Dans le code, une solution est un dictionnaire dont les clés sont les machines et les valeurs sont les listes ordonnées d'opérations traitées par chaque machine avec les dates de début et de fin correspondantes. De plus, 5 instances sont mises à votre disposition.

- `factory_A_6_6.txt` d'optimum connu égal à 47. Cette instance ne rapporte aucun point et est donnée pour réaliser vos tests.
- `factory_B_10_5.txt` d'optimum connu égal à 502.
- `factory_C_10_10.txt` d'optimum connu égal à 756.
- `factory_D_15_10.txt` d'optimum inconnu.
- `factory_E_15_15.txt` d'optimum inconnu.

Votre code sera aussi évalué sur une instance cachée (`factory_X.txt`), de taille légèrement supérieure à la dernière. Pour vérifier que tout fonctionne bien, vous pouvez exécuter le solveur naïf comme suit.

```
python3 main.py --agent=naive --infile=instances/factory_A_6_6.txt
```

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 : *optimisation par colonie de fourmis* (ACO), *recherche locale itérée* (ILS), *recherche à voisinage variable* (VNS), *recherche à voisinage large* (LNS), ou *optimisation par essaims de particules* (PSO). Vous êtes libres de choisir la métaheuristique de votre choix. Comme pour les devoirs précédents, réflé-

chissez 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. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=instances/factory_A_6_6.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 conception, et reporter 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. L'instance B rapporte 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 en diminuant 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).
— 2 points bonus seront attribués au groupe ayant implémenté l'approche la plus recherchée (adéquation des mécanismes intégrés dans l'algorithme).

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

Conseils

Voici quelques conseils pour le mener le devoir à bien :

1. Veillez à bien prendre en compte les commentaires faits pour les premiers devoirs.
2. Tirez le meilleur parti des séances de laboratoire encadrées afin de demander des conseils.
3. Inspirez vous des techniques vues au cours, et ajoutez-y vos propres idées.
4. 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.
5. 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 `solutionS` où S est le nom de l'instance
- Votre rapport de présentation de votre méthode et de vos résultats, d'au maximum 2 pages.